CRITICAL PROBLEMS IN VERY LARGE SCALE COMPUTER SYSTEMS

Semiannual Technical Report for the period April 1, 1988 to September 30, 1988

Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Principal Investigators:		
	Paul Penfield, Jr.	(617) 253-2506
	Anant Agarwal	(617) 253-1448
	William J. Dally	(617) 253-6043
	Srinivas Devadas	(617) 253-045+
	Thomas F. Knight, Jr.	(617) 253-7807
	F. Thomson Leighton	(617) 253-3662
	Charles E. Leiserson	(617) 253-5833
	Jacob K. White	(617) 253-2543
	John L. Wyatt, Jr.	(617) 253-6718

This research was sponsored by Defense Advanced Research Projects Agency (DoD), through the Office of Naval Research under Contract No. N00014-87-K-0825.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

TABLE OF CONTENTS

Selected Publications (starting after page 11)

*S. M. N. Hassoun, A Memory Design for the Message-Driven Processor, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1988. Also, MIT VLSI Memo No. 88-465, August 1988.

*Paul Y. Song, Design of a Network for Concurrent Message Passing Systems, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1988. Also, MIT VLSI Memo No. 88-458, August 1988.

*K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Finding the Steady State Solution of Clocked Analog Circuits," 1988 Custom Integrated Circuits Conference, Rochester, NY, May 16-18, 1988. Also, MIT VLSI Memo No. 88-444, March, 1988.

*S. Fiske and W. J. Dally, "The Reconfigurable Arithmetic Processor," Proceedings, 15th International Symposium on Computer Architecture, Honolulu, HI, May 30-June 2, 1988. Also, MIT VLSI Memo No. 88-449, June 1988.

*D. Smart and J. White "Reducing the Parallel Solution Time of Sparse Circuit Matrices using Reordered Gaussian Elimination and Relaxation," Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-440, March, 1988.

M. Reichelt, J. White, J. Allen and F. Odeh, "Waveform Relaxation Applied to Transient Device Simulation," Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-441, March, 1988.

S. Bhatt, F. Chung, J. Hong, T. Leighton, and A. Rosenberg, "Optimal Simulations by Butterfly Networks," Proceedings 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, May 2-4, 1988. Extended abstract issued as MIT VLSI Memo No. 87-427, November, 1987. Final version issued as MIT VLSI Memo No. 88-469, September 1988.

*W. J. Dally, A. Chien, S. Fiske, W. Horwat, J. Keen, P. Nuth, J. Larivee, and B. Totty, "Message-Driven Processor Architecture," MIT VLSI Memo No. 88-468, August 1988.

*W. J. Dally and A. A. Chien, "Object-Oriented Concurrent Programming in CST," MIT VLSI Memo No. 88-450, June 1988.

W. J. Dally, "Micro-Optimization of Floating-Point Operations," MIT VLSI Memo No. 88-470, August 1988.

---

* Abstract only. Complete version available from Microsystems Research Center, Room 39-321, MIT, Cambridge, MA 02139; telephone (617) 253-8138.

RESEARCH OVERVIEW

The research vehicle for this contract is the largest possible computer that could be conceived for the mid to late 1990s. The technical challenges of such a machine serve as the guiding stimulus for the research carried out and reported here.

We imagine this machine to occupy a 14-story building, to cost upwards of $1,000,000,000, and to be so colossal that the nation can only afford one or two of them. The available chip technology and machine size are consistent with a million billion FLOPS (that's 10 to the 15th) and a million billion Bytes of memory. It will dissipate 50 megawatts of power using CMOS technology. Communication across the machine will be much slower than computation at a node. The architecture, software, interconnect technology, packaging, and operating system are unknown.

This investigation deals with hardware technology, software techniques, programming algorithms, communications, processing elements, and applications. The study will determine the plausibility (not feasibility) of such a machine. Progress in these various areas are highlighted in the individual sections below.

CIRCUITS

A returning faculty member, Prof. Thomas F. Knight, Jr., has taken over this aspect of the work following the departure of Prof. Lance A. Glasser. Some immediate plans for this activity are outlined below.

We will design and construct a low latency processor to processor communication switch which uses innovative ideas at the architectural, silicon, and packaging levels to reduce communications delays. At the architectural level, the impact of simple source-responsible routing protocols on the development of fault tolerant, extremely simple routing elements will be investigated. The reduction in complexity of the routing part contributes to its speed. At the circuit level, high speed chip to chip communication techniques such as transistor series terminated drivers will be investigated, as well as some ideas for using high speed microwave modems to communicate in a non-baseband environment. In packaging, the objective includes a liquid cooled, dense, three-dimensional second level wiring technology, with almost isotropic wire density in all three dimensions.

PROCESSING ELEMENTS

Prof. Dally and his students have made significant progress in development of processing elements and associated communications circuits.

We developed a deadlock-free adaptive routing algorithm for k-ary n-cube networks. This algorithm will be used to implement fault-tolerant networks.

We completed an analysis and simulation study of network performance using different flow control strategies. This study showed that adaptive routing does not significantly improve performance. A flow control discipline that permits messages to pass one another is needed to improve performance further.

In the laboratory we demonstrated a number of our network and arithmetic concepts in three prototype chips: the NDF router, the RAP arithmetic chip, and a high-bandwidth memory chip.

A computer the size of the American Resource Computer will require the ability to change state rapidly to hide transmission latency without sacrificing single-thread performance. We are working on an architecture for a named state processor that explicitly binds names to registers. This mechanism combines the advantages of multi-threading and multiple register sets for implementing fast context switches and procedure calls.

We are investigating programming strategies for very large numbers of processors based on agents and agencies (Minsky, Society of the Mind). We are planning to use agencies to implement concurrency abstractions for naming and information sharing.

We are investigating the application of a computer of the scale of the American Resource Computer to database applications. The issues involved include data partitioning, methods for insuring stability and persistence, concurrency control, and efficient algorithms for search and update.

COMMUNICATIONS TOPOLOGY AND ROUTING ALGORITHMS

Charles E. Leiserson is currently on leave at Thinking Machines Corporation, through December 1988. On his return to MIT, he plans to continue his investigations into parallel computation, focusing principally on issues related to timing, synchronization, fault tolerance, and routing algorithms. He also expects to complete a textbook, entitled Introduction to Algorithms, coauthored with Thomas H. Cormen and Ronald L. Rivest.

Guy Blelloch is finishing his thesis, titled "Scan Primitives and Parallel Vector Models." The thesis suggests a new class of algorithmic models for parallel computing. These models are based on a set of operations on vectors of atomic values. The thesis shows how the models can be used for algorithm design, how they can be implemented on various computers, and how they can be used as the back end of a compiler for high-level languages. The thesis also suggests that a set of scan operations should be considered primitive parallel operations. Next year he will be an Assistant Professor at Carnegie Mellon University.

Tom Cormen has concentrated on writing the textbook Introduction to Algorithms with Professors Leiserson and Rivest. The textbook includes several chapters on parallel algorithms and circuitry. The writing should be completed by the end of 1988.

Jeff Fried is currently working on a number of problems to the impact of synchrony on the performance of distributed algorithms. He is also working on the architecture and blocking analysis of sparse circuit-switched interconnection networks. His most significant results relate to the design of VLSI processors for use within the interconnection networks found in telecommunications, distributed computing, and parallel processing.

Ron Greenberg and Mike Foster of Columbia have established matching lower and upper bounds for the area required to implement finite-state machines in VLSI. In addition Greenberg has continued work on the subject of universal routing networks for parallel computation. Greenberg and Leiserson's paper on compact layout of the three-dimensional tree of meshes provides a simple proof of results used to establish upper bounds on the penalty paid for using a general network to simulate all parallel machines, and Greenberg is currently making progress on lower bound results.

Greenberg and Alexander Ishii have also been working with Alberto Sangiovanni-Vincentelli of Berkeley on a multi-layer channel router for VLSI circuits, called `MulCh`. While based on the `Chameleon` system developed at Berkeley, `MulCh` incorporates the additional feature that nets may be routed entirely on a single interconnect layer (`Chameleon` requires the vertical and horizontal sections of a net be routed on different interconnect layers). When used on sample problems, `MulCh` shows significant improvements over `Chameleon` in area, total wire length, and via count.

Ishii has completed his masters thesis, which describes his models for VLSI timing analysis. The model maps continuous data-domains, such as voltage, into discrete, or *digital*, data domains, while retaining a continuous notion of time. The majority of the thesis concentrates on developing lemmas and theorems that can serve as a set of "axioms" when analyzing algorithms based on the model. Key axioms include the fact that circuits in our model generate only well defined digital signals, and the fact that components in our model support and accurately handle the "undefined" values that electrical signals must take on when they make a transition between valid logic levels. In order to facilitate proofs for circuit properties, the class of *computational predicates* is defined. A circuit property can be proved by simply casting the property as a computational predicate.

Ishii has also been working with Bruce Maggs on a new VLSI design for a high-speed multi-port register file. Design goals include short cycle-time and single-cycle register window context changes. This research

began as an advanced VLSI class project, under the supervision of Prof. Knight of the MIT Artificial Intelligence Laboratory.

James K. Park is currently collaborating with Alok Aggarwal and Dina Kravets on a number of problems in Computational Geometry. He is also working with Bruce Maggs on the problem of finding an optimal offline deterministic routing algorithm for the butterfly-fat-tree. Park's most significant contribution of the past year was his work with Aggarwal on monotone arrays.

Cindy Phillips and Charles Leiserson extended the graph contraction results of Phillips to lead to $O(\lg n + \lg^2\gamma)$-time randomized algorithms for finding the connected components (and related problems) of n-node bounded degree graphs where γ is the maximum genus of any connected component. With Guy Blelloch and (from Thinking Machines Corp) Ajit Agrawal and Robert Krawitz, Phillips investigated primitives for efficiently manipulating dense matrices in massively parallel hypercube architectures where many matrix elements must be mapped to a single processor.

Serge A. Plotkin finished his Ph.D. thesis entitled Graph-Theoretic Techniques for Parallel, Distributed, and Sequential Computation. His thesis includes the following results:

- A novel algorithm for symmetry breaking in distributed and parallel computing environments that runs in $O(\lg^* n)$ time.

- A new atomic data object, called a *Sticky Bit*. A polynomial number of Sticky Bits are sufficient to convert a safe, implementation of an arbitrary sequential object into an atomic one in a shared-memory multiprocessing environment.

- An algorithm for managing a global resource in a distributed network. In particular, the algorithm allows a resource used by a protocol of n processors to be managed with only amortized $O(\lg^2 n)$ message overhead.

- A parallel algorithm for solving the minimum spanning tree problem on a n-by-n mesh-connected computer that runs in $O(n)$ time. The algorithm is novel because it is based on reducing the minimum spanning tree problem to the problem of finding shortest paths.

- The first sublinear-time parallel algorithm for bipartite matching. The algorithm runs in $O(n^{2/3}\lg^3 n)$ time on a graph of n vertices, and can be generalized to solve 0-1 flow problems, both including both weighted and unweighted versions.

- Two sequential algorithms for the generalized circulation problem (network flow with losses and gains) which are the first polynomial-time combinatorial algorithms for this problem. One algorithm runs in $O(n^2m^2\lg^2 n \lg B)$ time and the other runs in $O(n^2m^2\lg n \lg^2 B)$ time, where n is the number of nodes, m is the number of edges, and B is the largest integer used to represent capacities and gains, where gains are represented as ratios of integers.

Plotkin has assumed a postdoctoral position at Stanford University.

Mark Newman's interests include fault tolerant parallel computation and efficient procedures for simulating one parallel network with another. During the past year, he completed work with Leighton and Johan Hastad which showed how a hypercube with a large number of faulty processors and communication paths could be used for computation. They showed that, even if a constant fraction of the hypercube's components fail, the cube can simulate a fully functioning hypercube using only a constant factor more time. In the next year, Newman plans to extend the results for faulty hypercubes to other networks and to search for efficient graph embeddings which aid in network simulation.

SYSTEMS SOFTWARE

Studies in scalability of large-scale shared-memory multiprocessors focussing on the use of locality in various forms to reduce the latency of memory accesses. A major part of the work headed by Prof. Anant Agarwal has also focussed on developing better data collection and evaluation techniques for multiprocessors.

The data from several address tracing techniques that we developed for both symbolic and numeric computing showed that parallel programs exhibit a significant amount of locality, and that this locality could be successfully exploited by caches at the processor level to provide a high effective memory bandwidth to the processor. An evaluation of the large-scale interconnection network performance of both hardware cache coherence (based on a novel directory structure) and software coherence schemes showed that the hardware directory scheme could perform well under significant sharing levels, while the software schemes could be relied upon for low to moderate sharing levels.

Our future research will focus on two aspects. Investigating high-performance interconnection technology for large-scale multiprocessing. We are building a prototype network clocked at 100MHz that will provide an average memory access time for a 256-processor system of less than 200ns. We are also investigating how locality of addressing can be incorporated into the network and to what extent programs can exploit the locality in the network. The second aspect of our research will research novel techniques of synchronization such a barriers and semaphores with a back-off capability to reduce network traffic by minimizing unnecessary spins on the network, and do a detailed design of the directory structure required to maintain cache coherence on a large scale. Plus, continued work on network and cache evaluation techniques and multiprocessor data collection efforts.

ALGORITHMS

Prof. Leighton and his students have discovered a very efficient randomized algorithm for routing in such networks as the hypercube, butterfly and shuffle-exchange graph that is robust in the sense that the same algorithm works for virtually any network in near-optimal time (e.g., even in arrays).

They have also discovered an entire class of approximation algorithms for layout related problems in VLSI such as graph partitioning, crossing number and layout area.

In addition, they have discovered efficient embeddings for a variety of useful networks in the hypercube and butterfly. Such embeddings are useful for mapping processes to processors in both synchronous and asynchronous parallel machines.

Michelangelo Grigni drafted "Tight Bounds on Minimum Broadcast Networks" with David Peleg of the Weizmann Institute (previously Stanford).

A certain class of recursively structured graphs had been proposed an examples of graphs which required small wire area, but large chip area, to lay out. Mark Hansen disproved this conjecture and demonstrated that this class of graphs have chip layout area equal to wire area. He has also developed some techniques for proving lower bounds on the area required to embed rectangular grids in square grids.

Richard Koch has probabilistically analyzed a routing scheme which has been implemented on parallel architectures based on the butterfly graph.

Dina Kravets developed algorithms for finding all farthest neighbors of every vertex on a convex n-gon in $\Theta(n)$ time, for sorting every row of a monotone matrix in $\Theta(n^2)$ time, and for sorting a set of numbers given ranges of ranks in $\Theta(n \log Q/n)$ where Q is the sum of the ranges.

Satish Rao and Tom Leighton have found the first approximation algorithms for the problems of finding small graph separators, VLSI layout, and crossing number. Leighton, Maggs, and Rao have explored solutions to packet routing problems with fixed congestion and dilation in LMR. They show the existence of a constant overhead schedule for such problem.

Eric Schwabe proved a general lower bound showing that any bounded-degree network which can manage m local priority-queue memories must have total size $\Omega(m \log m)$, even if randomized algorithms are allowed. This lower bound can be achieved -- meaning it is a network and algorithm which can manage m such memories in $O(m \log m)$ total space. As a side result of the techniques used in this algorithm, Hansen developed a simple algorithm for permutation routing of n messages on a butterfly network deterministically and on-line in $\Theta((\log^2 n)/(\log \log n))$ steps.

APPLICATIONS

Profs. Jacob White and Srinivas Devadas and their students are investigating difficult simulation tasks in an effort to challenge the capabilities of the American Resource Computer.

In this past year we proved a result about the optimality of Gauss-Jacobi over Gauss-Seidel on parallel processors, and developed a banded Gauss-Jacobi relaxation approach to simulating circuits that is fast and reliable. In addition, we proved several new results about the uniformity of WR convergence for nonlinear diagonally dominant systems, and demonstrated the result's practical implications on one dimensional MOS device simulation. We also reformulated the capacitance extraction problem into an iterative algorithm whose steps involve a potential field from point charge calculation, for which order N log N approximate algorithms exist. This implies that it is possible to reduce the complexity of capacitance extraction problem from the commonly used N^3 approach to N log N. Finally, we found several new approaches for the detailed simulation of switching filter circuits, and in particular implemented a new method for distortion calculation of switched capacitor filters.

In the immediate future, we will continue the investigation of the capacitance calculation problem, and will also try to apply the N log N approach to the calculation of inductances. In the area of device simulation, we will be working on numerical techniques for solving the hydrodynamics-based MOS device equations, parallel iterative techniques for two and three dimensional device simulation, and parallel WR for mixed device-circuit simulation. In the area of circuit simulation, we are investigating parallel nonlinear multi-grid like techniques for the simulation of analog arrays, investigating parallel exponential fitting discretization schemes, and trying to extend the approach to simulation of clocked analog circuits to phase-locked loops.

PUBLICATIONS LIST

R. I. Greenberg and C. E. Leiserson, "A Compact Layout for the Three-Dimensional Tree of Meshes," Applied Mathematics Letters, Vol. 1(2), pp. 171--176, 1988.

S. M. N. Hassoun, A Memory Design for the Message-Driven Processor, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1988. Also, MIT VLSI Memo No. 88-465, August 1988.

Paul Y. Song, Design of a Network for Concurrent Message Passing Systems, M. S. Thesis, Department of Electrical Engineering and Computer Science, MIT, May 1988. Also, MIT VLSI Memo No. 88-458, August 1988.

Seth Malitz, Optimal and Near Optimal Results on Book Embeddings," Ph.D. Thesis, Department of Mathematics, Massachusetts Institute of Technology, September, 1988.

S. A. Plotkin, Graph-Theoretic Techniques for Parallel, Distributed, and Sequential Computation, Ph.D. Thesis, September 1988.

K. Kundert, J. White, and A. Sangiovanni-Vincentelli, "A Mixed Frequency-Time Approach for Finding the Steady State Solution of Clocked Analog Circuits," 1988 Custom Integrated Circuits Conference, Rochester, NY, May 16-18, 1988. Also, MIT VLSI Memo No. 88-444, March, 1988.

S. Fiske and W. J. Dally, "The Reconfigurable Arithmetic Processor," Proceedings, 15th International Symposium on Computer Architecture, Honolulu, HI, May 30-June 2, 1988. Also, MIT VLSI Memo No. 88-449, June 1988.

P. R. O'Brien, J. L. Wyatt, Jr., T. L. Savarino, and J. M. Pierce, "Fast On-Chip Delay Estimation for Cell-Based Emitter Coupled Logic," Proceedings, 1988 IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also MIT VLSI Memo No. 88-436, February, 1988.

J. L. Wyatt, Jr., and D. L. Standley, "Circuit Design Criteria for Stable Lateral Inhibition Neural Networks," Proceedings, 1988 IEEE International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-439, March, 1988.

D. Smart and J. White "Reducing the Parallel Solution Time of Sparse Circuit Matrices using Reordered Gaussian Elimination and Relaxation," Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-440, March, 1988.

D. L. Standley and J. L. Wyatt, Jr., "Stability Theorem for Lateral Inhibition Networks that is Robust in the Presence of Circuit Parasitics," Proceedings of the IEEE International Conference on Neural Networks, San Diego, CA., July 1988, vol. I, pp. 27-36; also International Neural Networks Society 1st Annual Meeting, Boston, MA, September 1988. (Abstract appeared on p. 135 of Advance Program.)

M. Reichelt, J. White, J. Allen and F. Odeh, "Waveform Relaxation Applied to Transient Device Simulation," Proceedings, 1988 International Symposium on Circuits and Systems, Espoo, Finland, June 7-9, 1988. Also, MIT VLSI Memo No. 88-441, March, 1988.

S. Bhatt, F. Chung, J. Hong, T. Leighton, and A. Rosenberg, "Optimal Simulations by Butterfly Networks," Proceedings 20th Annual ACM Symposium on Theory of Computing, Chicago, IL, May 2-4, 1988. Extended abstract issued as MIT VLSI Memo No. 87-427, November, 1987. Final version issued as MIT VLSI Memo No. 88-469, September 1988.

L. Finkelstein, D. Kleitman, and F. T. Leighton, "Applying the Classification Theorem for Finite Simple Groups to Minimize Pin Count in Uniform Permutation Architectures," Proceedings of the Aegean Workshop on Computing, Corfu, Greece, June 27 - July 1, 1988, pp. 247-256.

F. T. Leighton, "A survey of Bounds and Algorithms for Channel Routing," Proceedings of the Bonn Workshop on Paths, Flows and VLSI Layout, Bonn, West Germany, June 20 - June 24, 1988.

J. Fried, "Optical Interconnections for VLSI," Proceedings of the Binghamton Photonics Symposium, Binghamton, NY, May 1988.

G. E. Blelloch and J. J. Little, "Parallel Solutions to Geometric Problems on the of Computation," Proceedings International Conference on Parallel Processing, Saint Charles, IL, August 15-19, 1988.

J. L. Wyatt, Jr., "Design Methodology for Stabilizing Lateral Inhibition Networks Coupled Through Resistive Grids," Conference on Neural Nets for Computing, Snowbird, UT, April 1988.

A. Agarwal and D. Sites, "Multiprocessor Cache Analysis Using ATUM," 15th International Symposium on Computer Architecture, Honolulu, Hawaii, June 1988.

A. Agarwal, R. Simoni, M. Horowitz, and J. Hennessy, "An Evaluation of Directory Schemes for Cache Coherence," 15th International Symposium on Computer Architecture, Honolulu, Hawaii, June 1988.

INTERNAL MEMORANDA

W. J. Dally, A. Chien, S. Fiske, W. Horwat, J. Keen, P. Nuth, J. Larivee, and B. Totty, "Message-Driven Processor Architecture," MIT VLSI Memo No. 88-468, August 1988.

W. J. Dally, "Micro-Optimization of Floating-Point Operations," MIT VLSI Memo No. 88-470, August 1988.

S. Owicki and A. Agarwal, "Evaluating the performance of software cache coherence," August 31, 1988. Submitted to ASPLOS-III.

S. Bhatt, F. Chung, F. T. Leighton and A. Rosenberg, "Universal Graphs for Bounded-Degree Trees and Planar Graphs," manuscript, June 1988.

S. Bhatt, F. Chung, F. T. Leighton and A. Rosenberg, "Efficient Embeddings of Trees in Hypercubes," manuscript, July 1988.

W. J. Dally and A. A. Chien, "Object-Oriented Concurrent Programming in CST," MIT VLSI Memo No. 88-450, June 1988.

M. Grigni and D. Peleg, "Tight Bounds on Minimum Broadcast Networks," to be submitted.

F. T. Leighton, "A 2d-1 Lower Bound for Knock-Knee Channel Routing," manuscript, September 1988.

F. T. Leighton, F. Makedon and I. Tollis, "A (2n-2)--Step Algorithm for Routing in an n*n Array," manuscript, July, 1988.

K. Kundert, J. White, A. Sangiovanni-Vincentelli. "A Mixed Frequency-Time Approach for Distortion Analysis of Switched Capacitor Filters." Submitted to the IEEE Journal of Solid State Circuits.

R. Saleh, J. White, A. Sangiovanni-Vincentelli, and A. R. Newton, "Accelerating Relaxation Algorithms for Circuit Simulation Using Waveform-Newton and Step-Size Refinement," Submitted to the IEEE Transactions on Computer-Aided Design.

TALKS WITHOUT PROCEEDINGS

F. T. Leighton, "Flows, Paths and VLSI Layout" Bonn Workshop on Flows, Paths and VLSI Layout, Bonn, West Germany, June 20-24, 1988.

F. T. Leighton, "Groups, Pins and Chips," Aegean Workshop on Computing, Corfu, Greece, June 27 - July 1, 1988.

J. White, "Parallel Circuit Simulation Algorithms," Design Automation Conference, Anaheim, CA, June 1988.

A. Agarwal, "Can We Build Large-Scale Shared-Memory Multiprocessors?" IBM T. J. Watson Research Center, Yorktown Heights, NY, July 11, 1988.

A. Agarwal and A. Gupta, "Memory Reference Characteristics of Multiprocessor Applications under MACH" ACM SIGMETRICS, Santa Fe, NM, June 1988.

F. T. Leighton, "Survey Talk on Networks, Parallel Computation and VLSI Design," presented at the Trento School on VLSI Computation, 6-lecture series, July 1988, and EATCS International College on Automata, Languages and Programming, plenary lecture, July 1988.

F. T. Leighton, "Some Computational Properties of the Hypercube," presented at Carnegie-Mellon University, Pittsburgh, PA, April 1988 and at Pennsylvania State University, University Park, PA, April 1988.

F. T. Leighton, "Universal Algorithms for Packet Routing, IDA Supercomputing Research Center, Washington, DC, June 1988.

VLSI Memo No. 88-465
August 1988

A MEMORY DESIGN FOR THE MESSAGE-DRIVEN PROCESSOR

Soha M. N. Hassoun

Abstract

The Message-Driven Processor (MDP) is a low-latency processing node for a scalable fine-grain MIMD concurrent computer, the Jellybean Machine. Programs are executed by passing messages through a low-latency network. Each MDP integrates a processor, a memory, and a communication network. On top of this message-passing model, the MDP supports a global virtual address space.

This thesis involves the design and implementation of a memory for the Message-Driven Processor. The memory array can be accessed by index, by row, or as a set-associative cache. Index operations are used to read and write memory. Row operations reduce the latency in message-handling by providing special purpose buffers, Row Buffers that access four words (a row) of memory simultaneously. Two Queue Row Buffers enable buffering messages at two different priority levels as soon as they arrive from the network. An Instruction Row Buffer acts as a small instruction cache. Set-associative operations provide a translation mechanism to enable translating any object to its associated item. MDP operating system routines use this cache to translate virtual identifiers into global addresses.

The microarchitecture and the circuit design of the memory is developed. A test chip is fabricated to verify the design. Evaluation of the row operations is presented.

Microsystems Research Center Room 39-321
Massachusetts Institute of Technology
Cambridge Massachusetts 02139
Telephone (617) 253-8138

VLSI Memo No. 88-458
August 1988

DESIGN OF A NETWORK FOR CONCURRENT MESSAGE PASSING SYSTEMS

Paul Y. Song

Abstract

We describe the design of the network design frame (NDF), a self-timed routing chip for a message-passing concurrent computer. The NDF uses a partitioned data path, low-voltage output drivers, and a distributed token-passing arbiter to provide a bandwidth of 450 Mbits/sec into the network. Wormhole routing and bidirectional virtual channels are used to provide low latency communications, less than 2us latency to deliver a 216 bit message across the diameter of a 1K node mess-connected machine. To support concurrent software systems, the NDF provides two logical networks, one for user messages and one for system messages. The two networks share the same set of physical wires. To facilitate the development of network nodes, the NDF is a design frame. The NDF circuitry is integrated into the pad frame of a chip leaving the center of the chip uncommitted.

We define an analytic framework in which to study the effects of network size, network buffering capacity, bidirectional channels, and traffic on this class of networks. The response of the network to various combinations of these parameters are obtained through extensive simulation of the network model. Through simulation, we are able to observe the macro behavior of the network as opposed to the micro behavior of the NDF routing controller.

We subsequently define the limitations of the network and propose recommendations for enhancing the network performance. The limitation of the network arises from contention for the switching elements of the NDF. The use of virtual channels allows better utilization of network bandwidth by doubling the number of switches at each node. A three dimensional version of the NDF will be needed to support large machines that exceed 16 nodes in a dimension. Adding a third dimension increases the bisection width of the network and gives us more throughput.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

VLSI Memo No. 88-444
March 1988

A MIXED FREQUENCY-TIME APPROACH FOR FINDING THE STEADY-STATE SOLUTION OF CLOCKED ANALOG CIRCUITS

K. Kundert, J. White, and A. Sangiovanni-Vincentelli

Abstract

Performing detailed simulation of clocked analog circuits (e.g. switched-capacitor filters and switching power supplies) with circuit simulation programs like SPICE is computationally very expensive. In this paper we present a new, more efficient, method for computing the detailed steady-state solution of clocked analog circuits. The method exploits the property of such circuits that the waveforms in each clock cycle are similar but not exact duplicates of the proceeding or following cycles. Therefore, by computing accurately a few selected cycles, the entire steady-state solution can be constructed efficiently.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

Massachusetts Institute of Technology | **Microsystems Research Center** | Cambridge Massachusetts 02139 | Room 39-321 Telephone (617) 253-8138

VLSI Memo No. 88-449
June 1988

THE RECONFIGURABLE ARITHMETIC PROCESSOR

Stuart Fiske and William J. Dally

Abstract

The Reconfigurable Arithmetic Processor (RAP) is an arithmetic processing node for a message-passing, MIMD concurrent computer. It incorporates on one chip several serial, 64 bit floating point arithmetic units connected by a switching network. By sequencing the switch through different patterns, the RAP chip calculates complete arithmetic formulas. By chaining together its arithmetic units the RAP reduces the amount of off chip data transfer; In the examples we have simulated off chip I/O can often be reduced to 30% or 40% of that required by a conventional arithmetic chip. Simulations predict a peak performance of 20M Flops with 800M bit/sec off chip bandwidth in a 2 μm CMOS process.

VLSI Memo No. 88-440
March 1988

REDUCING THE PARALLEL SOLUTION TIME OF SPARSE CIRCUIT MATRICES USING REORDERED GAUSSIAN ELIMINATION AND RELAXATION

David Smart and Jacob White

Abstract

Using parallel processors to reduce the execution times of classical circuit simulation programs like SPICE and ASTAP has been the focus of much current research. In these efforts, good parallel speed increases have been achieved for linearized system construction, but it has been difficult to get good parallel speed increases for sparse matrix solution. In this paper we examine two approaches for reducing parallel sparse matrix solution time; the first based on pivot ordering algorithms for Gaussian elimination, and the second based on relaxation algorithms. In the section on Gaussian elimination sparse matrix solution, we present a pivot ordering algorithm which increases the parallelism of Gaussian elimination compared to the commonly used Markowitz method. The performance of the new algorithm is compared to other suggested ordering algorithms for a collection of circuit examples. The minimum number of parallel steps for the solution of a tridiagonal matrix is derived, and it is shown that this optimum is nearly achieved by the ordering heuristics which attempt to maximize parallelism. In the section on relaxation, we present an optimality result about Gauss-Jacobi over Gauss-Seidel relaxation on parallel processors.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

VLSI Memo No. 88-441
March 1988

WAVEFORM RELAXATION APPLIED TO TRANSIENT DEVICE SIMULATION

M. Reichelt, J. White, J. Allen, and F. Odeh

Abstract

In this paper we investigate the possibility of accelerating the transient simulation of MOS devices by using waveform relaxation. Standard spatial discretization techniques are used to generate a large, sparsely-connected system of algebraic and ordinary differential equations in time. The waveform relaxation (WR) algorithm for solving such a system is described, and several theoretical results that characterize the convergence of WR for device simulation are given. In addition, one-dimensional experimental results are presented.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency contract number N00014-87-K-0825, and the Air Force Office of Scientific Research grant number AFOSR-86-0164.

Author Information

Reichelt, White, and Allen: Research Laboratory of Electronics, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA 02139; Reichelt: Room 36-897, (617) 253-1204; White: Room 36-880, (617) 253-2543; Allen: Room 36-413, (617) 253-3103; Odeh: IBM T.J. Watson Research Center, Yorktown Heights, NY, (914) 945-3000.

Waveform Relaxation Applied to Transient Device Simulation

M. Reichelt, J. White, J. Allen
Research Laboratory of Electronics and the
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA
F. Odeh
I.B.M. T. J. Watson Reseach Center
Yorktown Heights, NY

Abstract

In this paper we investigate the possibility of accelerating the transient simulation of MOS devices by using waveform relaxation. Standard spatial discretization techniques are used to generate a large, sparsely-connected system of algebraic and ordinary differential equations in time. The waveform relaxation (WR) algorithm for solving such a system is described, and several theoretical results that characterize the convergence of WR for device simulation are given. In addition, one-dimensional experimental results are presented.

1 Introduction

Both digital and analog MOS circuit designers rely heavily on circuit simulation programs like SPICE [3] to insure the correctness and to test the performance of their designs. For most applications, the lumped MOS models used in these programs [9] accurately reflect the behavior of terminal currents and charges, but in some cases, these models are not adequate. In particular, charge redistribution between source and drain during device switching cannot easily be modeled by a lumped device, but the details of this charge redistribution can have an important effect on circuit behavior. In circuits like dynamic memory cells, sense amplifiers, analog-to-digital converters, and high frequency operational amplifiers, charge redistribution effects may not only degrade performance, but can inhibit proper function.

For these critical applications, sufficiently accurate transient simulations can be performed if, instead of using a lumped model for each transistor, the transis-

tor terminal currents and charges are computed by numerically solving the drift-diffusion based partial differential equation approximation for electron transport in the device. However, simulating even a few transistor circuit in this way is very computationally expensive, because the accurate solution of the transport transport equations an MOS device requires a two dimensional mesh with more than a thousand points.

In this paper we investigate the possibility of accelerating the transient simulation of MOS devices by using waveform relaxation. In the next section we start by introducing the equations for transient device simulation. Then we view the result of applying commonly used spatial discretization techniques to these equations, generating a large, sparsely-connected system consisting of algebraic and ordinary differential equations in time. In Section 3 we present the waveform relaxation algorithm for solving such a system, and suggest why it may be particularly efficient. Several theoretical results that characterize the convergence of the method are presented in Section 4, and one-dimensional experimental results are described in section 5. Finally, conclusions and acknowledgements are given in section 6.

2 Classical Simulation Equations

The terminal behavior of an MOS device is well described by the Poisson equation and the electron current-continuity equation [5]

$$\epsilon \nabla^2 \psi + q\,(N - n) = 0 \tag{1}$$

$$\nabla \cdot \vec{J}_n - q\frac{\partial n}{\partial t} = 0 \tag{2}$$

In these equations ψ is the electrostatic potential, q is the magnitude of electronic charge, n is the electron concentration, and $\vec{J}_n$ is the electron current density. N is the net doping concentration given by $N = N_D - N_A$ where N_D and N_A are the donor and acceptor concentrations.

The electron current density is commonly approximated by the drift-diffusion equation:

$$\vec{J}_n = -q\,(\mu_n\, n\, \nabla\psi - D_n \nabla n) \tag{3}$$

where μ_n is the electron mobility, and D_n is the diffusion coefficient. An equation system with only n and ψ as unknowns is derived by using (3) to eliminate $\vec{J}_n$ from (2).

There are a variety of ways to spatially discretize the system of two equations in the two unknowns n and ψ. Given a rectangular two dimensional mesh, a common approach is to use a finite-difference formula for the Poisson equation, and an exponentially-fit finite-difference formula for the current-continuity equation. For notational simplicity, we will assume that the mesh points are

evenly spaced a distance l apart, so that the discretized Poisson equation at each mesh point i is:

$$\epsilon \sum_j (\psi_j - \psi_i) + ql^2 (N_i - n_i) = 0 \tag{4}$$

where n_i, ψ_i, and N_i are the electron concentration, the potential, and the net doping concentration at mesh point i. The summation is taken over the nodes j surrounding i (four nodes for a mesh node i not on the boundary, i.e. north, south, east, and west).

Under the same assumptions, and assuming constant mobility, the discretized current-continuity equation with the drift-diffusion approximation becomes:

$$qD_n \sum_j [B(u_j - u_i)n_j - B(u_i - u_j)n_i] - ql^2 \left(\frac{d}{dt} n_i\right) = 0 \tag{5}$$

where $u_i = q\psi_i/KT$ and $B(x) = x/(\exp x - 1)$ is the Bernoulli function used to exponentially fit the potential variation to the electron concentration variation. In this equation, the Einstein relation $D_n = (KT/q)\mu_n$ has been used to eliminate μ_n.

If there are m mesh points, then the result of applying the spatial discretization to (1),(2), and (3) is a sparse system of m algebraic constraints, represented by (4), and a sparsely connected system of m ordinary differential equations, represented by (5).

3 The Waveform Relaxation Process

The standard approach used to solve these two systems is to discretize the $\frac{d}{dt} n_i(t)$ term in (5) with a low order integration method such as backward-Euler [1]. The result is a sequence of algebraic systems in $2m$ unknowns, each of which can be solved with some variant of Newton's method and/or relaxation. Another approach is to apply relaxation directly to the differential equation system. This leads to a time waveform relaxation process, as given by the following algorithm.

Although only the Gauss-Jacobi algorithm is presented for the sake of notational simplicity, a Gauss-Seidel version could be created by adjusting the iteration indexes.

The WR algorithm reduces the problem of simultaneously solving m differential equations and m algebraic equations to one of iteratively solving $2m$ independent equations. Each of the m differential equations for the $n_i(t)$ waveforms can be solved with a numerical integration method such as backward-Euler. Since they only contribute algebraic constraints, the equations for calculating the $\psi_i(t)$ waveforms need to be solved only at the discrete points in time used to calculate the $n_i(t)$ waveforms.

Algorithm 1 WR Gauss-Jacobi Algorithm for solving the system produced by equations (4) and (5).

The superscript k denotes the iteration count, the subscript i denotes the component index of a vector, and ϵ_ψ and ϵ_n are small positive numbers.

$k \leftarrow 0$
repeat {
  $k \leftarrow k + 1$
  foreach$(i \in \{1, \ldots, n\})$ {
    solve

$$\epsilon \sum \left(\psi_j^{k-1} - \psi_i^k\right) + ql^2 \left(N_i - n_i^{k-1}\right) = 0$$

$$qD_n \sum \left[B(u_j^{k-1} - u_i^{k-1})n_j^{k-1} - B(u_i^{k-1} - u_j^{k-1})n_i^k\right]$$

$$-ql^2 \left(\tfrac{d}{di} n_i^k\right) = 0$$

    for$(\psi_i^k(t), n_i^k(t);\ t \in [0, T], n_i^k(0) = n_{i0})$
  }
} **until**$(\|\psi^k - \psi^{k-1}\| \leq \epsilon_\psi$ **and** $\|n^k - n^{k-1}\| \leq \epsilon_n)$

The inherent advantage of the WR approach is that the differential equations are solved in a decomposed fashion, and therefore different sets of timesteps can be used at different mesh points to calculate the time evolution of the electron concentration. The method exploits multi-rate behavior. In MOS devices, the rate at which electron concentrations evolve may be very different in the channel compared to the source or the drain. Therefore, WR may prove to be efficient for the device simulation problem, provided it converges, and doesn't take too many iterations. This is the subject of the next section.

4 Theoretical Results

As is usually the case for waveform relaxation algorithms applied to systems of differential equations, Algorithm 1 converges to the solution of the differential-algebraic system for any initial guess that matches the initial conditions. The precise statement is given in the following theorem.

Theorem 1 *Given a finite interval* $[0,T]$, *and any initial guess* $n^0(t)$ *and* $\psi^0(t)$, $t \in [0,T]$, *such that* $n^0(0) = n_0$, *the sequence of waveforms produced by Alg. 1 converges to the exact solution of the system given by equations (4) and (5).*

The proof of the above theorem follows the same steps as the Picard-like proofs of waveform relaxation for ordinary differential equations [10]. First the equations that describe the difference between one iteration and the next are organized into the form

$$\delta\psi^{k+1} = A\delta\psi^k + B\delta n^k(t) \tag{6}$$

and

$$\delta n^{k+1}(t) = \int_0^t \left[f(n^{k+1}(t), n^k(t), \psi^k(t)) - f(n^k(t), n^{k-1}(t), \psi^{k-1}(t))\right] \tag{7}$$

where $\delta\psi_i^k = \psi_i^k - \psi_i^{k-1}$, $\delta n_i^k = n_i^k - n_i^{k-1}$. The matrices $A, B \in \Re^{m\times m}$ and the function $f : \Re^{m\times m\times m} \rightarrow \Re^m$ are constructed from the iteration equations in Alg. 1. The next step is to show that (6) and (7) represent a contraction. To this end, consider an interval of time short enough to insure equation (7) represents a contraction with respect to n for a fixed ψ. That (6) is a contraction with respect to ψ for a fixed n is well-known [8], as (6) represents relaxation applied to the Poisson equation. One can fit the two contractions together to show that relaxation applied to the coupled system converges.

The above proof outline suggests that the WR algorithm converges in a nonuniform manner. That is, first convergence is achieved over a short time interval, set by what is needed to make (7) a contraction, then over the next short time interval, and then the next, continuing slowly, until the convergence is achieved throughout an entire interval of interest. When applied to general

differential equation systems, like circuits, WR does demonstrate this nonuniformity in the convergence [7], but WR does not usually show nonuniformity when applied to the transient device simulation problem.

In order to analyze why this is the case, we will consider a model problem of just the differential equation associated with the electron concentration, n and assume that the potential ψ is known. The WR iteration update equation for this case is then

$$D_n \sum_j \left[B(u_j - u_i) n_j^k - B(u_i - u_j) n_i^{k+1} \right] - l^2 \left(\frac{d}{dt} n_i^{k+1} \right) = 0 \qquad (8)$$

for each $i \in \{1, \ldots m\}$. Note that given ψ, (8) is a linear time-varying differential equation in n. For this problem we have the following theorem:

Theorem 2 *If at each time t, $\psi(t)$ is such that the electric field along any vertical or horizontal line is either constant, or monotonically increasing, then (8) is a contraction in a uniform norm on any finite interval $[0, T]$. That is,*

$$max_{[0,T]} ||\delta n^{k+1}(t)|| \leq \gamma max_{[0,T]} ||\delta n^k(t)|| \qquad (9)$$

where $\gamma < 1$.

The proof of Theorem 2 is given in the appendix.

Since allowing the different differential equations to take very different timesteps is WR's main advantage, if this property were limited to insure convergence, the WR algorithm would not be effective. Fortunately, that the WR algorithm is a contraction in a unform norm on any interval implies that the timesteps used to numerically integrate the differential equations are almost unconstrained. Given that the different differential equations use different timesteps, interpolation must be used to communicate results between equations, and if not done carefully this can cause nonconvergence. Linear interpolation is certain not cause problems, and therefore we have the following theorem [7]:

Theorem 3 *Let each of the m independent WR iteration update equations given in (8) be solved numerically with backward-Euler, with m different sets of timesteps. In addition, assume that linear interpolation is used to derive values for the n_i's between time discretization points. Then this multirate discretized WR algorithm for (8) converges, regardless of the timestep selections.*

5 One Dimensional Experiments

Except for Theorem 1, the above theoretical results only apply under certain conditions, and are only an indication that the WR algorithm may be effective. In order to verify that the theoretical results apply in actual simulation, a one-dimensional transient device simulation program was written and applied to a

one-dimensional approximation of an MOS device with a conducting channel. The doping distribution for the one-dimensional device is given in Fig. 1, where the tick marks denote the mesh points. Potential and electron concentration boundary conditions were given at $x = 0.0$ and $x = 3.0\mu$. The boundary values for the electron concentration were computed assuming charge neutrality at the "contacts".

The relaxation process was tested by first solving the static problem with zero volts across the "device", and then making a step change of five volts. Even with this simple example, the variable-by-variable WR algorithm as given in Alg. 1 was ineffective. The iterates did not converge in a uniform manner, and they converged very slowly.

In order to improve convergence, rather than using variable-by-variable decomposition, we partitioned the problem into blocks based on two techniques. First, we associated the electron concentration at node i, $n_i(t)$ with the potential $\psi_i(t)$ at that node. Then, in order to try to satisfy the assumptions of Theorem 2, we placed together neighboring nodes where we expected rapid changes in the electric field. The resulting partitioning of the nodes are boxed in Fig. 1.

The resulting waveform iterations for the slowest converging variable, the electron concentration for the mesh point where the doping changes abruptly, is plotted in Fig. 2. As the figure indicates, with the partitioning just described, the WR process converges in just a few iterations and the contraction is uniform through time as predicted by Theorem 2. The simulation was rerun with very coarse timesteps to see the effects on convergence, and the WR iterations for the same node is plotted in Fig. 3. As the figure indicates, using coarse timesteps does not effect the overall convergence, although the convergence for small t is slowed.

6 Conclusions and Acknowledgements

In this paper we presented some preliminary results that indicate the WR algorithm may indeed be efficient for device transient simulation. In particular, it was shown that under conditions that can be arranged for in practice, the WR algorithm is a contraction in a uniform norm on any interval $[0, T]$. Also, given these same conditions, the relaxation process will still converge even if very different sets of timesteps are used for the individual iteration equations. Finally, we verified the theoretical results on a one dimensional example.

There are several aspects of WR that need to be addressed if this method it to be efficent for two-dimensional MOS transient device simulation. Most important, a general algorithm for blocking the device must be developed. An efficent approach for determining what discretization points to use for the algebraic constraints must be considered. In addition, the efficiency of WR methods can also be improved by refining the timesteps with iterations, or using a single waveform-Newton iteration to solve the nonlinear WR iteration equations.

The authors would like to thank W. Van Bokhoven for suggesting this research area and John Wyatt for his suggestion on the proof of theorem 2. This work was supported by the Defense Advanced Research Projects Agency contract N00014-87-K-825, and the Air Force Office of Scientific Research grant AFOSR-86-0164.

References

[1] R. E. Bank, et al., "Transient Simulation of Silicon Devices and Circuits", *IEEE Trans. Electron Dev.*, vol ED-32 no. 10, 1985, pp. 1992-2007.

[2] U. Miekkala and O. Nevanlinna, "Convergence of Dynamic Iteration Methods for Initial Value Problems", *Report-MAT-A230*, Institute of Mathematics, Helsinki University of Technology, Finland, 1985.

[3] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits", Electronic Research Laboratory *Report No. ERL-M520*, University of California, Berkeley, May 1975.

[4] C. S. Rafferty, M. R. Pinto, and R. W. Dutton, "Iterative Methods in Semiconductor Device Simulation", *IEEE Trans. Electron Dev.*, vol ED-32 no. 10, 1985, pp. 2018-2027.

[5] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*, Springer-Verlag, New York, 1984.

[6] M. Vidyasagar, *Nonlinear Systems Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[7] J. K. White and A. L. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of VLSI Circuits*, Kluwer Academic Publishers, Boston, 1987.

[8] Richard S. Varga, *Matrix Iterative Analysis*, Prentice Hall, Englewood Cliffs, New Jersey, 1962.

[9] Richard S. Muller and Theodore I. Kamins *Device Electronics for Integrated Circuits* John Wiley and Sons, New York, 1987.

[10] E. Lelarasmee, A. Ruehli, A. Sangiovanni-Vincentelli, *The Waveform Relaxation Method for the Time Domain Analysis of Large Scale Integrated Circuits*, IEEE Trans. on CAD, Vol. 1, No. 3, July 1982.

A Proof of Theorem 2

The WR iteration equations applied to the model problem (8) can be described as

$$\dot{n}^{k+1}(t) = D(t)n^{k+1}(t) + M(t)n^k(t) \tag{10}$$

where $D(t), M(t) \in \Re^{n\times n}$, and $D(t)$ is negative diagonal matrix. The assumptions about the electric field result in values for the Bernoulli functions such that $D(t)$ and $M(t)$ will satisfy the relation

$$\|d_{ii}(t)\| \geq \epsilon_i + \sum_{j\neq i} \|m_{ij}(t)\|. \tag{11}$$

where $\epsilon_i \geq 0$ and is strictly greater than zero for those $i's$ corresponding to the mesh points next to the boundaries. Note that this implies

$$\|D(t)^{-1}M(t)\| \leq \gamma \tag{12}$$

for $\gamma < 1$, for some norm on $\Re^{n\times n}$ and for all t.

Given the relationship between $D(t)$ and $M(t)$, the WR algorithm applied to a system of the form of (13) will contract in a uniform norm. This has been shown for the case when $D(t)$ and $M(t)$ are independent of t, using Laplace transforms [2]. In the time dependent case, the result can be shown by examining the difference between iteration k and $k+1$ of (13) to get

$$\dot{\delta n}_i^{k+1}(t) = d_i i(t)\delta n_i^{k+1}(t) + \sum_{i\neq j} m_{ij}(t)\delta n_j^k(t) \tag{13}$$

for each mesh point i, where $\delta n_i^k(t) = n_i^k(t) - n_i^{k-1}(t)$. By assumption, $d_{ii}(t) < 0$ and $\delta n_i^k(0) = 0$. Therefore,

$$max_{[0,T]}|\delta n_i^{k+1}(t)| \leq \sum_{i\neq j} max_{[0,T]}|\frac{m_{ij}(t)}{d_{ii}(t)}|max_{[0,T]}|\delta n_j^k(t)|. \tag{14}$$

Equation (14) follows from the fact that for all values of $\delta n^{k+1}(t)$ on the boundary of (or outside) the bounded region $\dot{\delta n}_i^{k+1}(t)$ points back into the bounded region [6].

Assembling the equation system from (14) results in

$$max_{[0,T]}|\delta n^{k+1}(t)| \leq max_{[0,T]}|D(t)^{-1}M(t)|max_{[0,T]}|\delta n^k(t)|. \tag{15}$$

Then in the norm for which $\|D(t)^{-1}M(t)\| \leq \gamma < 1.0$,

$$max_{[0,T]}\|\delta n^{k+1}(t)\| \leq \gamma max_{[0,T]}\|\delta n^k(t)\|. \tag{16}$$

which proves the theorem.

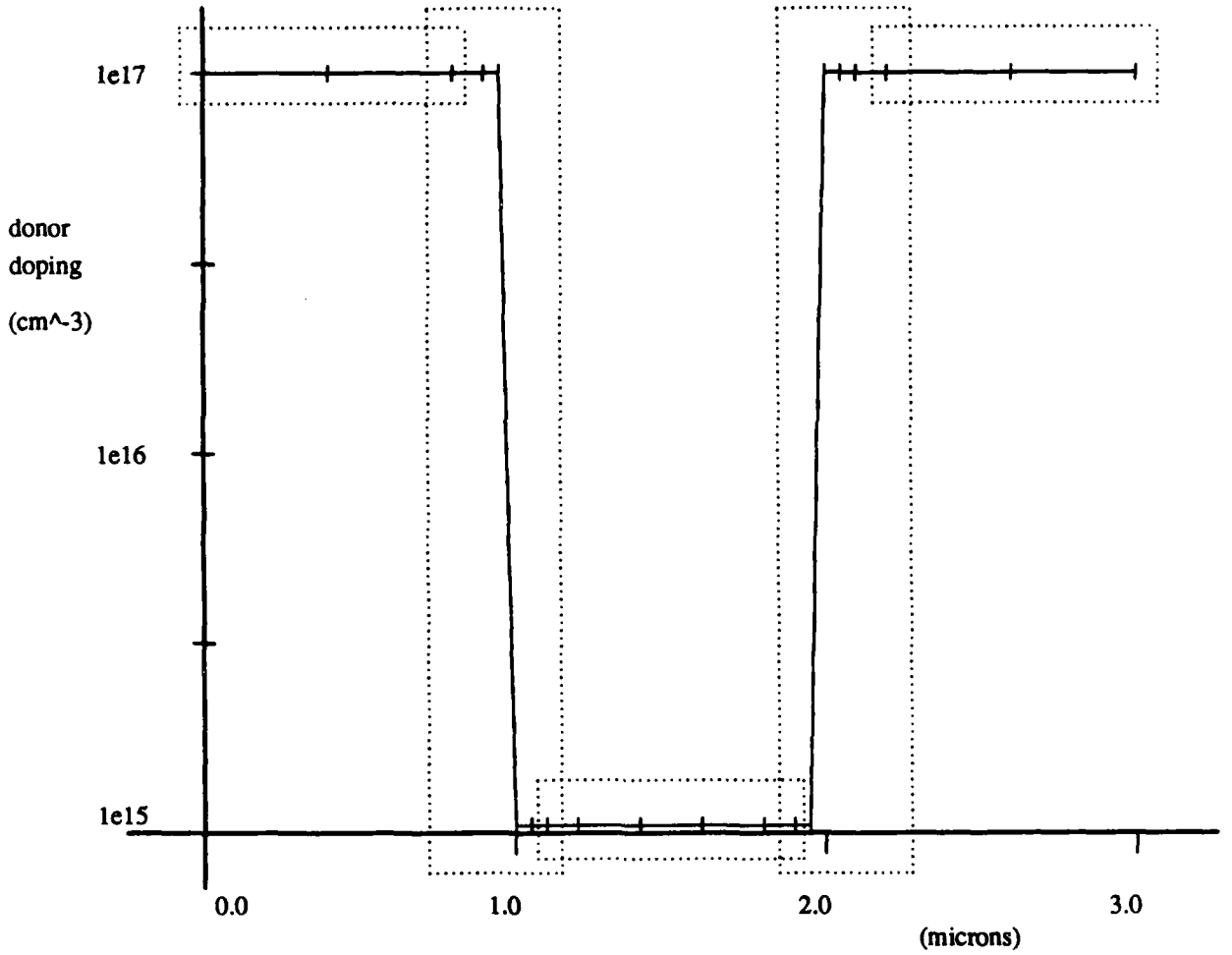


Figure 1: The net doping profile, and the blocking of the mesh points.

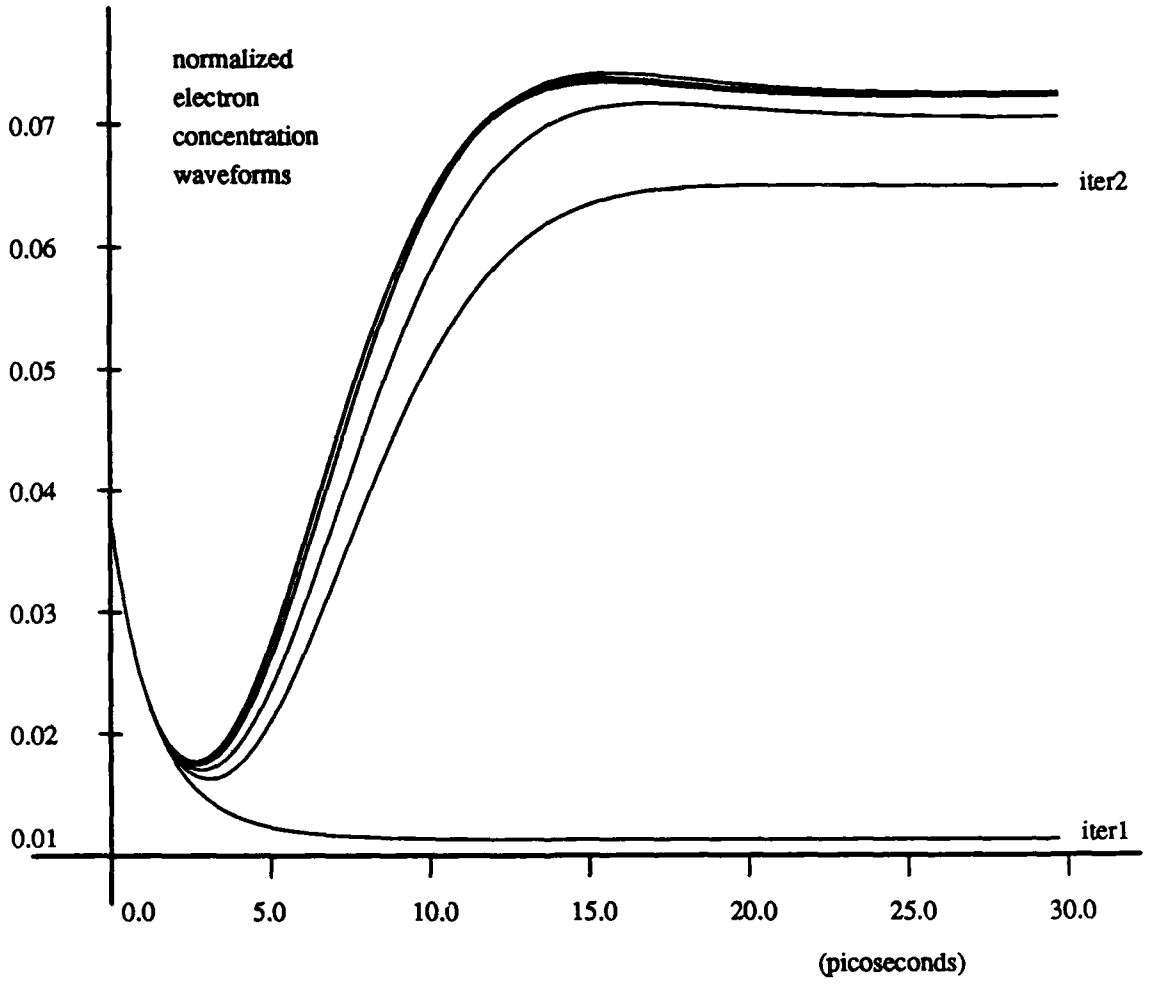


Figure 2: The uniform WR convergence of the electron concentration at a node.

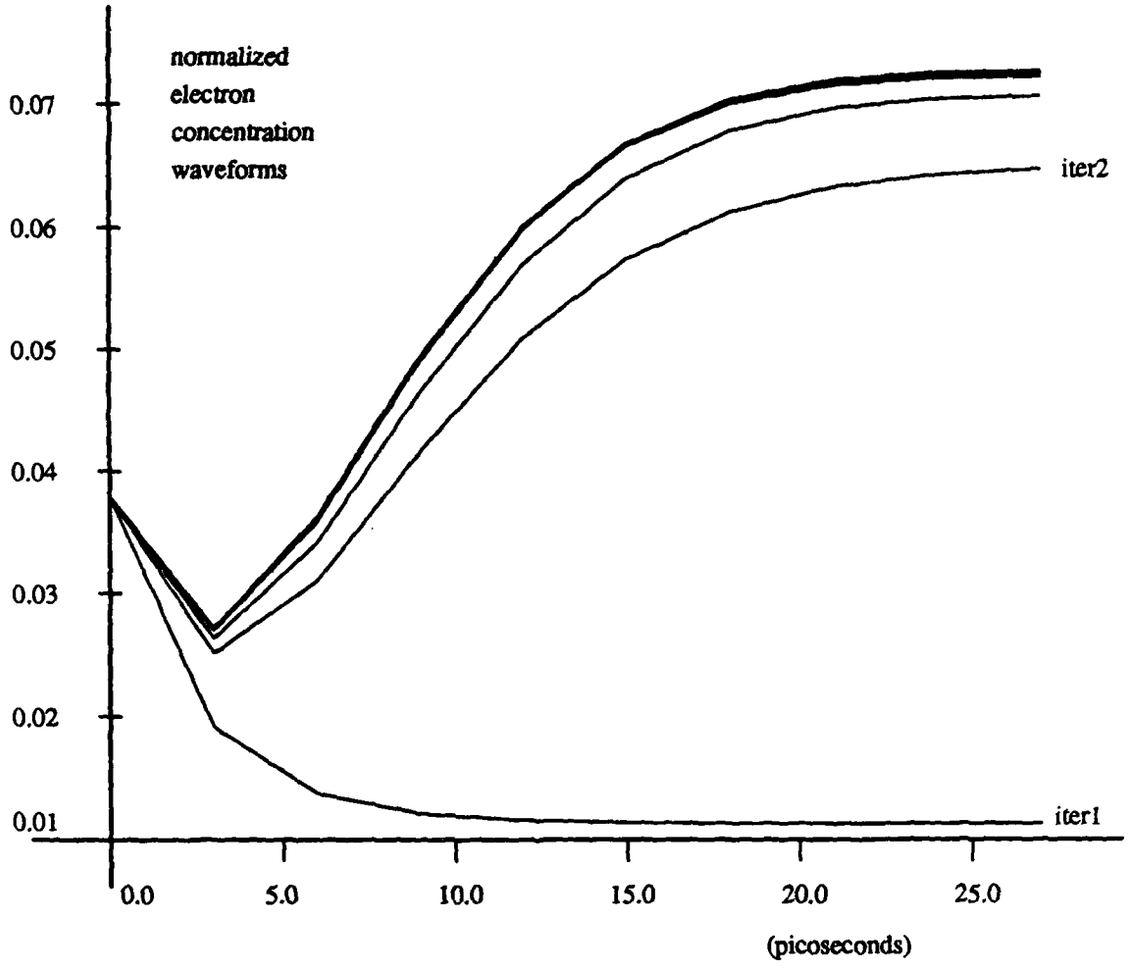


Figure 3: The waveforms converge uniformly, even when the timesteps are coarse.

VLSI Memo No. 88-469
August 1988

OPTIMAL SIMULATIONS BY BUTTERFLY NETWORKS

Sandeep N. Bhatt, Fan R. K. Chung, Jia-Wei Hong, F. Thomson Leighton, and Arnold L. Rosenberg

Abstract

The power of Butterfly-type networks relative to other proposed multicomputer interconnection networks is studied, by considering how efficiently the Butterfly can simulate the other networks. Simulation is represented formally via graph embeddings, so the topic here becomes: How efficiently can one embed the graph underlying a given network in the graph underlying the Butterfly network? The efficiency of an embedding of a graph G in a graph H is measured in terms of: the *dilation*, or, the maximum amount that any edge of G is "stretched" by the embedding; the *expansion*, or, the ratio of the number of vertices of H to the number of vertices of G. Three general results about embeddings in Butterfly-type graphs are established here, that expose a number of simulations by Butterfly-type networks, which are optimal (to within constant factors): (1) Any complete binary tree can be embedded in a Butterfly graph, with simultaneous dilation $O(1)$ and expansion $O(1)$. (2) Any n-vertex graph having a $\sqrt{2}$-*bifurcator* of size $S = \Omega(\log n)$ can be embedded in a Butterfly graph with simultaneous dilation $O(\log S)$ and expansion $O(1)$. (3) Any embedding of a planar graph G in a Butterfly graph must have dilation $\Omega\{[\log \Sigma(G)]/\Phi(G)\}$: $\Sigma(G)$ is the size of the smallest 1/3-2/3 vertex-separator of G; $\Phi(G)$ is the size of G's largest interior face. Corollaries include: (a) The n-vertex X-tree can be embedded in the Butterfly with simultaneous dilation $O(\log \log n)$ and expansion $O(1)$; no embedding yields smaller dilation, independent of expansion. (b) Every embedding of the $n \times n$ mesh in the Butterfly has dilation $\Omega(\log n)$; any expansion-$O(1)$ embedding of the mesh in the Butterfly achieves this dilation. These results, which extend to Butterfly-like graphs such as the Cube-Connected Cycles and Benes networks, supply the first examples of graphs that can be embedded more efficiently in the Hypercube than in the Butterfly.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge
Massachusetts
02139

Telephone
(617) 253-8138

Acknowledgements

Presented at the 20th ACM Symposium on Theory of Computing, Chicago, IL, May 2-4, 1988. This work was supported in part by NSF Grant Nos. MIP-86-01885, DCI-85-04308, and DCI-87-96236, Air Force Contract OSR-86-0076, the Defense Advanced Research Projects Agency under contract nos. N00014-80-C-0622 and N00014-87-K-0825, an NSF Presidential Young Investigators Award, with matching funds from IBM and AT&T.

Author Information

Bhatt: Department of Computer Science, Yale University, New Haven, CT 06520; Chung: Mathematics, Information Sciences and Operations Research Division, Bell Communications Research, Morristown, NJ 07960; Hong: Beijing Computer Institute, Beijing 10044, CHINA; Leighton: Department of Mathematics, MIT, Cambridge, MA 02139; Rosenberg: Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

OPTIMAL SIMULATIONS BY BUTTERFLY NETWORKS [1]

Sandeep N. Bhatt†, Fan R.K. Chung§,
Jia-Wei Hong‡, F. Thomson Leighton¶,
Arnold L. Rosenberg
Computer and Information Science Department
University of Massachusetts

†Yale University, New Haven CT
§Bell Communications Research, Morristown, NJ
‡Beijing Computer Institute, Beijing, CHINA
¶MIT, Cambridge, MA

[1] A preliminary version of this paper was presented at the *20th ACM Symposium on Theory of Computing*, Chicago, IL, May 2-4, 1988

Contact Author:
Arnold L. Rosenberg
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Acknowledgments of Support:
The research of S. N. Bhatt was supported in part by NSF Grant MIP-86-01885; the research of F.T. Leighton was supported in part by Air Force Contract OSR-86-0076, DARPA Contract N00014-80-C-0622, Army Contract DAAL-03-86-K-0171, and and NSF Presidential Young Investigator Award with matching funds from ATT and IBM; the research of A. L. Rosenberg was supported in part by NSF Grants DCI-85-04308 and DCI-87-96236.

Authors' Present Addresses:
Sandeep N. Bhatt: Department of Computer Science, Yale University, New Haven, CT 06520;
Fan R. K. Chung: Mathematics, Information Sciences and Operations Research Division, Bell Communications Research, Morristown, NJ 07960;
Jia-Wei Hong: Beijing Computer Institute, Beijing 10044, CHINA; currently visiting at Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003;
F. Thomson Leighton: Department of Mathematics, MIT, Cambridge, MA 02139;
Arnold L. Rosenberg: Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures – *interconnection architectures*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems – *Computations on discrete structures*; G.2.1 [**Discrete Mathematics**]: Combinatorics – *combinatorial algorithms*; G.2.2 [**Discrete Mathematics**]: Graph Theory – *graph algorithms*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Mapping algorithms, mapping problems, parallel architectures, processor arrays, simulation

Abstract

The power of Butterfly-type networks relative to other proposed multicomputer interconnection networks is studied, by considering how efficiently the Butterfly can simulate the other networks. Simulation is represented formally via graph embeddings, so the topic here becomes: How efficiently can one embed the graph underlying a given network in the graph underlying the Butterfly network? The efficiency of an embedding of a graph G in a graph H is measured in terms of: the *dilation*, or, the maximum amount that any edge of G is "stretched" by the embedding; the *expansion*, or, the ratio of the number of vertices of H to the number of vertices of G. Three general results about embeddings in Butterfly-type graphs are established here, that expose a number of simulations by Butterfly-type networks, which are optimal (to within constant factors): (1) Any complete binary tree can be embedded in a Butterfly graph, with simultaneous dilation $O(1)$ and expansion $O(1)$. (2) Any n-vertex graph having a $\sqrt{2}$-bifurcator of size $S = \Omega(\log n)$ can be embedded in a Butterfly graph with simultaneous dilation $O(\log S)$ and expansion $O(1)$. (3) Any embedding of a planar graph G in a Butterfly graph must have dilation $\Omega\left(\frac{\log \Sigma(G)}{\Phi(G)}\right)$: $\Sigma(G)$ is the size of the smallest 1/3-2/3 vertex-separator of G; $\Phi(G)$ is the size of G's largest interior face. Corollaries include: (a) The n-vertex X-tree can be embedded in the Butterfly with simultaneous dilation $O(\log \log n)$ and expansion $O(1)$; no embedding yields smaller dilation, independent of expansion. (b) Every embedding of the $n \times n$ mesh in the Butterfly has dilation $\Omega(\log n)$; any expansion-$O(1)$ embedding of the mesh in the Butterfly achieves this dilation. These results, which extend to Butterfly-like graphs such as the Cube-Connected Cycles and Benes networks, supply the first examples of graphs that can be embedded more efficiently in the Hypercube than in the Butterfly.

1. INTRODUCTION

This paper reports on a continuing program of the authors, dedicated to determining the relative computational capabilities of the various interconnection networks that have been proposed for use as multicomputer interconnection networks [BCLR, BI, GHR, Le]. We focus here on one member of the family of *butterfly*-like machines, that have become one of the benchmark architectures for multicomputers. The major contributions of this paper are the following general results about embeddings of graphs in Butterfly networks[1]:

1. We embed the complete binary tree in the Butterfly network, with simultaneous dilation $O(1)$ and expansion $O(1)$.
2. We embed any n-vertex graph having a $\sqrt{2}$-bifurcator of size $S = \Omega(\log n)$ in the Butterfly network, with simultaneous dilation $O(\log S)$ and expansion $O(1)$.
3. We prove that any embedding of any planar graph G in a Butterfly network must have dilation
$$\Omega\left(\frac{\log \Sigma(G)}{\Phi(G)}\right)$$
where: $\Sigma(G)$ is the size of the smallest 1/3-2/3 vertex-separator of G; $\Phi(G)$ is the size of G's largest interior face.

The latter two results lead to embeddings of graphs such as X-trees and meshes in the Butterfly, that are optimal, to within constant factors. By Result 2, such embeddings can be found with expansion $O(1)$ and with, respectively, dilation $O(\log \log n)$ and $O(\log n)$; by Result 3, no embeddings can improve on these dilations, independent of expansion. These embeddings expose X-trees and meshes as the *first known graphs that can be embedded very efficiently in the Hypercube* (simultaneous dilation $O(1)$ and expansion $O(1)$) *but have no efficient embedding in butterfly-like graphs.* Note that, if we restrict attention only to the issue of dilation, then – to within constant factors – these graphs cannot be embedded any more efficiently in Butterfly graphs than they can in complete binary trees!

1.1. The Formal Setting

The technical vehicle for our investigations is the following notion of graph embedding [Ro]. Let G and H be simple undirected graphs. An *embedding* of G in H is a

[1] All technical terms are defined in Section 1.1.

one-to-one association of the vertices of G with vertices of H, plus a *routing* of each edge of G within H, i.e., an assignment of a path in H connecting the images of the endpoints of each edge of G. The *dilation* of the embedding is the length of the longest path in H that routes an edge of G; it thus measures how much the edges of G are "stretched" by the embedding. The *expansion* of the embedding is the ratio $|H|/|G|$ of the number of vertices in $H$to the number of vertices in G. We use the dilation- and expansion-costs of the best embedding of G in H as our measures of how well H can *simulate* G as an interconnection network: One views the graph H as abstracting the processor-intercommunication structure of a physical architecture; one views the graph G as abstracting either the task-interdependency structure of an algorithm one wants to implement on H or the processor-intercommunication structure of an architecture one wants to simulate on H.

Remark. A third important measure of how well H can simulate G is *congestion*, the maximum number of edges that are routed through a single edge (or vertex) of H. Congestion does not play a major role in this paper, however, since
1. our embedding of a complete binary tree in a Butterfly trivially has unit congestion;
2. the n-vertex Butterfly is known to be able to simulate any n-vertex bounded-degree graph with $O(\log n)$ delay, irrespective of the fact that the dilation and congestion of the corresponding embedding may both be $\Omega(\log n)$; 3. our major focus is on developing broadly applicable techniques for bounding the dilation of embeddings.
Hence, for our purposes, dilation is the central measure of concern.

Our results hold for a large variety of "levelled" Hypercube-derivative host graphs (which play the role of our H's), that we collectively term *butterfly* networks. For the sake of rigor, we focus on one particular such network (which can be viewed as the FFT network, with input and output vertices identified), although we could just as easily substitute other such graphs - the Cube-Connected Cycles [PV] or Benes network [Be], for example. Formally,

- Let m be a positive integer. The *m-level Butterfly graph* $B(m)$ has vertex-set[2]

$$V_m = \{0, 1, \cdots, m-1\} \times \{0,1\}^m.$$

  The subset $V_{m,\ell} = \{\ell\} \times \{0,1\}^m$ of V_m $(0 \leq \ell < m)$ is the ℓ^{th} *level* of $B(m)$. The string $x \in \{0,1\}^m$ of vertex $\langle \ell, x \rangle$ is the *position-within-level string* (*PWL string*, for short) of the vertex. The edges of $B(m)$ form *butterflies* (or, copies

[2] $\{0,1\}^m$ denotes the set of length-m binary strings.

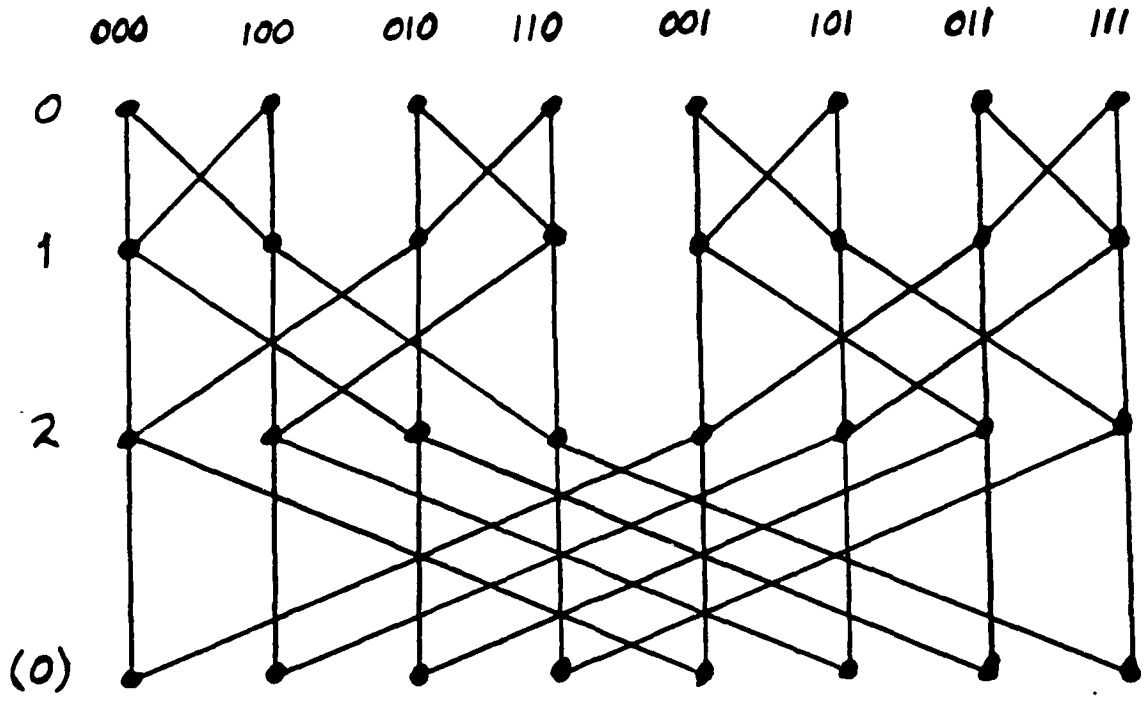


Figure 1: The 3-level Butterfly graph $B(3)$

of $K_{2,2}$) between consecutive levels of vertices, with wraparound in the sense that level 0 is identified with level m. Each butterfly connects vertices

$$\langle \ell,\ \beta_0\beta_1\cdots\beta_{\ell-1}0\beta_{\ell+1}\cdots\beta_{m-1}\rangle$$

and

$$\langle \ell,\ \beta_0\beta_1\cdots\beta_{\ell-1}1\beta_{\ell+1}\cdots\beta_{m-1}\rangle$$

on level ℓ of $B(m)$ $(0 \le \ell < m$; each $\beta_i \in \{0,1\})$ with vertices

$$\langle \ell+1(\text{mod } m),\ \beta_0\beta_1\cdots\beta_{\ell-1}0\beta_{\ell+1}\cdots\beta_{m-1}\rangle$$

and

$$\langle \ell+1(\text{mod } m),\ \beta_0\beta_1\cdots\beta_{\ell-1}1\beta_{\ell+1}\cdots\beta_{m-1}\rangle$$

on level $\ell+1(\text{mod } m)$ of $B(m)$. One can represent $B(m)$ level by level, in such a way that at each level the PWL strings are the reversals of the binary representations of the integers $0, 1, \ldots, 2^m-1$, in that order. See Fig. 1.

The guest graphs in our study, which play the role of our G's, are complete binary trees, X-trees, and meshes; see Fig. 2. Formally,

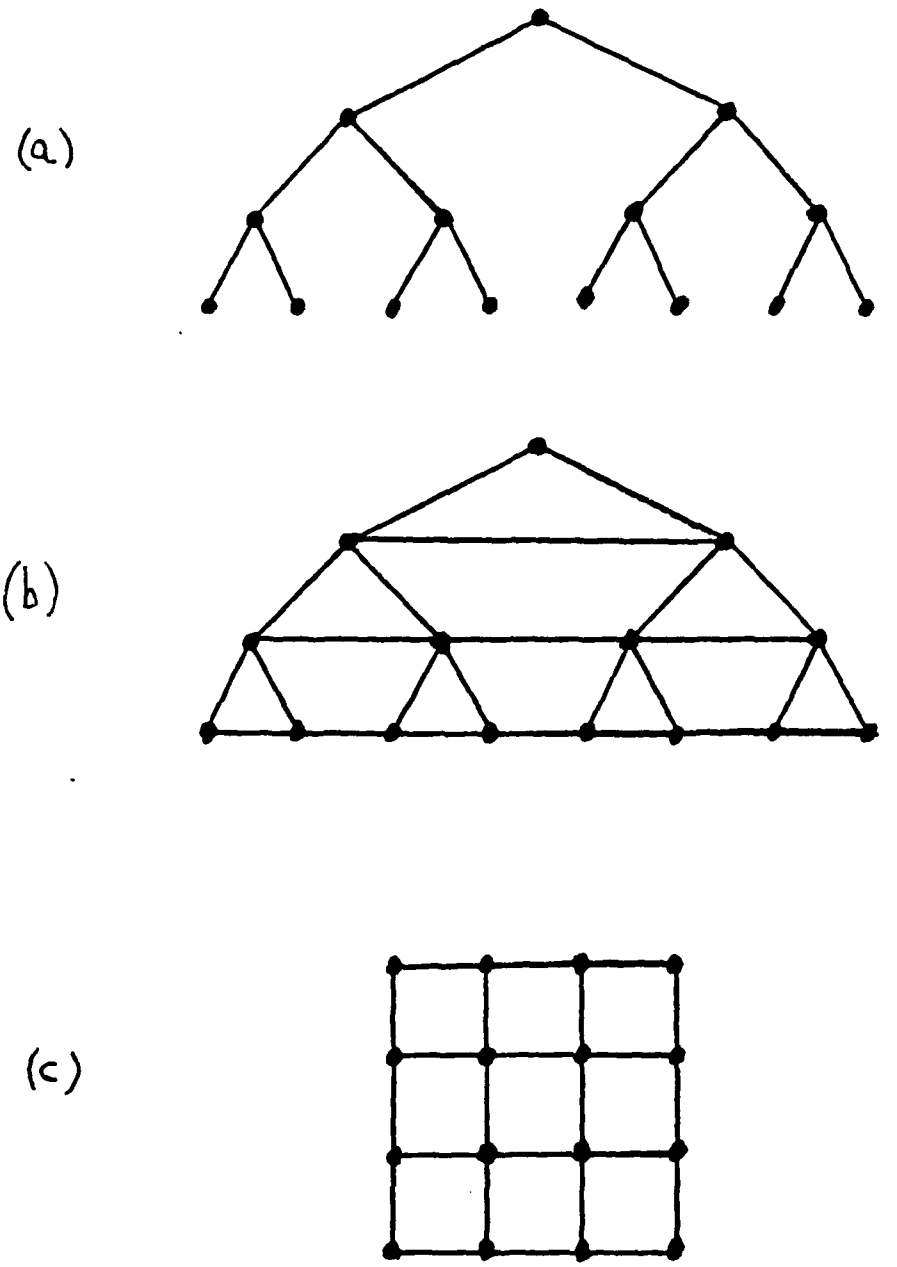


Figure 2: The Complete Binary tree $T(2)$, the X-tree $X(2)$, and the mesh $M(4)$

- The *height-h complete binary tree* $T(h)$ is the graph whose $(2^{h+1}-1)$-element vertex-set comprises all binary strings of length at most h, and whose edges connect each vertex x of length less than h with vertices $x0$ and $x1$. The (unique) string of length 0 is the *root* of the tree, which is the sole occupant of level 0 of the tree; the 2^ℓ strings of length ℓ are the *level-ℓ* vertices of the tree; the strings of length h (i.e., the level-h vertices) are the *leaves* of the tree.

- The *height-h X-tree* $X(h)$ is the graph that is obtained from the height-h complete binary tree $T(h)$ by adding *cross edges* connecting the vertices at each level of $T(h)$ in a path, with the vertices in lexicographic order. X-trees inherit a level structure from their underlying complete binary trees.

- The $s \times s$ *mesh* $M(s)$ is the graph whose s^2-element vertex set comprises the ordered pairs of integers

$$\{1, 2, \cdots, s\} \times \{1, 2, \cdots, s\},$$

and whose edges connect vertices $\langle a, b\rangle$ and $\langle c, d\rangle$ just when $|a-c|+|b-d| = 1$.

All of these networks have been seriously proposed as interconnection networks for multicomputers [DP, Ga, HZ], hence are important candidates for our study. Another approach to comparing these networks, via implementation and analysis of specific algorithms, appears in [Ag].

Our results depend on three structural features of a graph G:

1. Let S and k be positive integers. The n-vertex graph G has a *k-color $\sqrt{2}$-bifurcator of size S* if either $n < 2$ or the following holds for every way of labelling each vertex of G with one of k possible labels: By removing $\leq S$ vertices from G, one can partition G into subgraphs G_1 and G_2 such that[3]
   (a) $\big|\, |G_1| - |G_2| \,\big| \leq 1$.
   (b) For each label l, the number of l-labelled vertices in G_1 is within 1 of the number of l-labelled vertices in G_2.
   (c) Each of G_1 and G_2 has a k-color $\sqrt{2}$-bifurcator of size $S/\sqrt{2}$.

2. A *1/3-2/3 (vertex-)separator* of G is a set of vertices whose removal partitions G into subgraphs, each having $\geq |G|/3$ vertices; we denote by $\Sigma(G)$ the size of the smallest 1/3-2/3 vertex-separator of G.

3. When G is planar and we are given a witnessing planar embedding ϵ, we denote by $\Phi_\epsilon(G)$ the number of vertices in G's largest interior face in the embedding. When ϵ is clear from context, we omit the subscript.

[3] We denote by $|G|$ the number of vertices in the graph G.

1.2. The Main Results

We prove three results about optimal embeddings in the Butterfly that lead to a variety of nontrivial optimal embeddings.

Theorem 1 *The complete binary tree $T(h)$ can be embedded in a Butterfly graph, with simultaneous dilation $O(1)$ and expansion $O(1)$.*

Obviously, the embedding of Theorem 1 is within a constant factor of optimal in both dilation and expansion. Building on the embedding, we obtain the following general upper bound result.

Theorem 2 *Any n-vertex graph G having a $\sqrt{2}$-bifurcator of size $S = \Omega(\log n)$ can be embedded in a Butterfly graph with simultaneous dilation $O(\log S)$ and expansion $O(1)$.*

We balance Theorem 2 with one of the first broadly applicable results for bounding dilation from below.

Theorem 3 *Any embedding of a nontree planar graph G in a Butterfly graph has dilation $\Omega\left(\frac{\log \Sigma(G)}{\Phi(G)}\right)$. This bound cannot be improved in general.*

Direct application of the proofs of these results yields the following optimal embeddings.

Corollary 1 *The height-h X-tree $X(h)$ can be embedded in a Butterfly graph with simultaneous dilation $O(\log h) = O(\log \log |X(h)|)$ and expansion $O(1)$. Any embedding of $X(h)$ in a Butterfly graph must have dilation $\Omega(\log h) = \Omega(\log \log |X(h)|)$.*

Corollary 2 *Any embedding of the $s \times s$ mesh $M(s)$ in a Butterfly graph must have dilation $\Omega(\log s) = \Omega(\log |M(s)|)$.*

Corollary 2 betokens a mismatch in the structures of meshes and Butterfly graphs, since *any* expansion-$O(1)$ embedding of any graph G in $B(m)$ has dilation $O(\log |G|)$.[4]

[4] This follows from the facts that $B(m)$ has $m2^m$ vertices and diameter $O(m)$.

Theorem 1 and Corollaries 1 and 2 can be interpreted as yielding tight bounds on the efficiency with which a Butterfly machine can simulate a complete-binary-tree machine, an X-tree machine, and a mesh-structured machine, with regard to both delay (dilation) and resource utilization (expansion). Equating dilation with delay is most appropriate when the machines are to be run in SIMD mode.

The next three sections are devoted to proving our main results.

2. COMPLETE BINARY TREES

2.1. Embedding Many Small Trees in a Butterfly

It is obvious from inspection that one can find an instance of the height-$(m-1)$ complete binary tree $T(m-1)$ rooted at every vertex of $B(m)$. Somewhat less obvious is the fact that one can find m mutually disjoint instances of $T(m-1)$ as subgraphs of $B(m)$. We now verify this fact via an embedding which will prove useful as we develop our final embedding.

Proposition 1 *For every integer m, one can find m mutually disjoint instances of $T(m-1)$ as subgraphs of $B(m)$.*

Proof. To simplify exposition, we represent sets of binary strings by strings over the alphabet $\{0, 1, *\}$, using $*$ as a wild-card character. The length-k string

$$\beta = \beta_0\beta_1 \cdots \beta_{k-1},$$

where each $\beta_i \in \{0, 1, *\}$, represents the set $\sigma(\beta)$ of all length-k binary strings that have a 0 in each position i of β where $\beta_i = 0$, a 1 in each position i of β where $\beta_i = 1$, and either a 0 or a 1 in each position i of β where $\beta_i = *$. For illustration, $\sigma(010) = \{010\}$, and $\sigma(0 * 1) = \{001, 011\}$. Call the string β the *code* for the set $\sigma(\beta)$.

On to our embeddings of m instances of $T(m-1)$ in $B(m)$: For any letter a and nonnegative integer k, we denote by a^k a string of k a's.

For the first instance of $T(m-1)$, we have the following correspondence between tree vertices and Butterfly vertices.

$T(m-1)$	$B(m)$
level 0	$\langle 0,\ 0^m \rangle$
level 1	$\langle 1,\ *0^{m-1} \rangle$
level 2	$\langle 2,\ *^2 0^{m-2} \rangle$
$\vdots$	$\vdots$
level $m-1$	$\langle m-1,\ *^{m-1} 0 \rangle$

For each subsequent instance of $T(m-1)$, say the j^{th} where $1 < j < m$, we have the following correspondence between tree vertices and Butterfly vertices.

$T(m-1)$	$B(m)$
level 0	$\langle j-1,\ 0^{j-1}10^{m-j-1}1 \rangle$
level 1	$\langle j,\ 0^{j-1}1*0^{m-j-2}1 \rangle$
level 2	$\langle j+1 (\text{mod } m),\ 0^{j-1}1*^2 0^{m-j-3}1 \rangle$
$\vdots$	$\vdots$
level $m-1$	$\langle j-2,\ *^{j-1}1*^{m-j} \rangle$

The placement of the 1's in the PWL strings ensures that the m instances of $T(m-1)$ are mutually disjoint. To verify this, via contradiction, let us look at an arbitrary level ℓ of $B(m)$ and at arbitrary distinct tree vertices i and j that collide at some position within level ℓ of $B(m)$. It is clear that all Butterfly vertices that are images of the same instance of $T(m-1)$ are distinct, so we may assume that vertices i and j come from distinct instances of $T(m-1)$, call them $\iota(i)$ and $\iota(j)$, where the ι-"name" of an instance of $T(m-1)$ is the level of $B(m)$ where its root resides. We consider four cases that exhaust the possibilities. In each case, we adduce a property of the PWL strings that precludes any overlap in the images of the trees.

$\iota(i) = 0$:

> If $\iota(i) = 0$, then the PWL string of i ends with $0^{m-\ell}$, while the PWL string of j has a 1 in this range, specifically, in position $m-1$ if $j \leq \ell$, and in position j if $j > \ell$.

$1 \leq \iota(i) < \iota(j) \leq \ell$:

Every PWL string of $\iota(i)$ starts with 0^i1, while every PWL string of $\iota(j)$ starts with 0^j1.

$1 \leq \iota(i) \leq \ell < \iota(j)$:

Every PWL string of $\iota(i)$ has a 0 in position j, while every PWL string of $\iota(j)$ has a 1 in that position.

$\ell < \iota(i) < \iota(j) < m$:

Every PWL string of $\iota(i)$ has a 1 in position i, while every PWL string of $\iota(j)$ has a 0 in position i.

The proof is complete. □

An algebraic proof of Proposition 1, which is "cleaner" than our combinatorial proof here, appears in [ABR]; however, it is the embedding rather than the result that will be helpful in our proof of Theorem 1.

The embedding in our proof of Proposition 1 does not serve us directly in our attempt to embed a large complete binary tree in a small Butterfly, since (for one thing) it places the roots of every instance of $T(m-1)$ at a different level of $B(m)$; and it is not clear how to combine these instances into a bigger complete binary tree with small dilation. However, the overall strategy of the embedding will be useful in Section 2.2.*D*.

2.2. Optimally Embedding Trees in Butterfly Graphs

We turn now to the proof of Theorem 1. Specifically, we prove the following.

For any integer m, one can embed the complete binary tree $T(m + \lfloor \log m \rfloor - 1)$ in the Butterfly graph $B(m+3)$, with dilation $O(1)$.

To simplify our description, let $q =_{\text{def}} m + \lfloor \log m \rfloor - 1$, and assume henceforth that m is even; clerical changes will remove the assumption.

A. The Embedding Strategy

We wish to embed the tree $T(q)$ with dilation $O(1)$, in the smallest Butterfly that is big enough to hold the tree, namely, $B(m)$. We fall somewhat short of this

goal, but not by much: We find an embedding with dilation $O(1)$, but we have to use a somewhat larger host Butterfly graph (specifically, $B(m+3)$) in order to resolve collisions in our embedding procedure. Our embedding proceeds in four stages. Stage 1 embeds the top $\log m$ levels of $T(q)$ with unit dilation in $B(m)$, thereby specifying implicitly the images in $B(m)$ of the roots of the $m/2$ subtrees of $T(q)$ rooted at level $\log m - 1$. Stage 2 expands these subtrees a further $m/2$ levels, but now in $B(m+1)$, with dilation 2, thereby specifying implicitly the images in $B(m)$ of the roots of the $m \cdot 2^{m/2-1}$ subtrees of $T(q)$ rooted at level $m/2 + \log m - 1$ of the tree. In Stage 3, we embed the final $m/2$ levels of $T(q)$ in $B(m+1)$, with dilation 4. The vertex-mappings in each stage are embeddings (i.e., are one-to-one); there is, however, "overlap" (i.e., distinct vertices of $T(q)$ getting mapped to the same vertex of $B(m+1)$) among the mappings of the three stages. In Stage 4, we eliminate this overlap by expanding the host Butterfly by two more levels, thereby giving us four connected isomorphic copies of $B(m+1)$. At the cost of increasing dilation by 2, we modify our mapping so that each of Stages 1, 2, 3 is performed in a distinct copy of $B(m+1)$, thereby eliminating all overlap.

B. Stage 1: The Top $\log m$ Levels of $T(q)$

We place the root of $T(m + \log m)$ at position

$$\langle m - \log m,\ 0^m \rangle$$

of $B(m)$. We then proceed to higher-numbered levels, embedding the top $\log m$ levels of $T(q)$ as a subgraph of $B(m)$, ending up with the leaves of these levels in positions

$$\langle 0,\ 0^{m-\log m+1} *^{\log m-1} \rangle$$

of $B(m)$ (because of wraparound). See Fig. 3. We call the rightmost $\log m - 1$ bits of each of the resulting PWL strings the *signature* of the Butterfly position and of the subtree rooted at that position. It is convenient to interpret a signature as an integer in the range $\{0, 1, \cdots, m/2 - 1\}$, as well as a bit string.

The embedding in Stage 1 is trivially one-to-one, with unit dilation.

C. Stage 2: The Next $m/2$ Levels of $T(q)$

Call the $(m/2+1)$-level subtree of $T(q)$ that has signature k, the k^{th} *subtree*. Our goal is to embed the k^{th} subtree in $B(m+1)$ (with dilation 2), so that its $2^{m/2}$ leaves form the set of positions[5]

$$\langle m-1,\ *0*0\cdots*0*1*0\cdots*0*0? \rangle,$$

[5]The last bit position is not affected by this Stage, so is denoted "?".

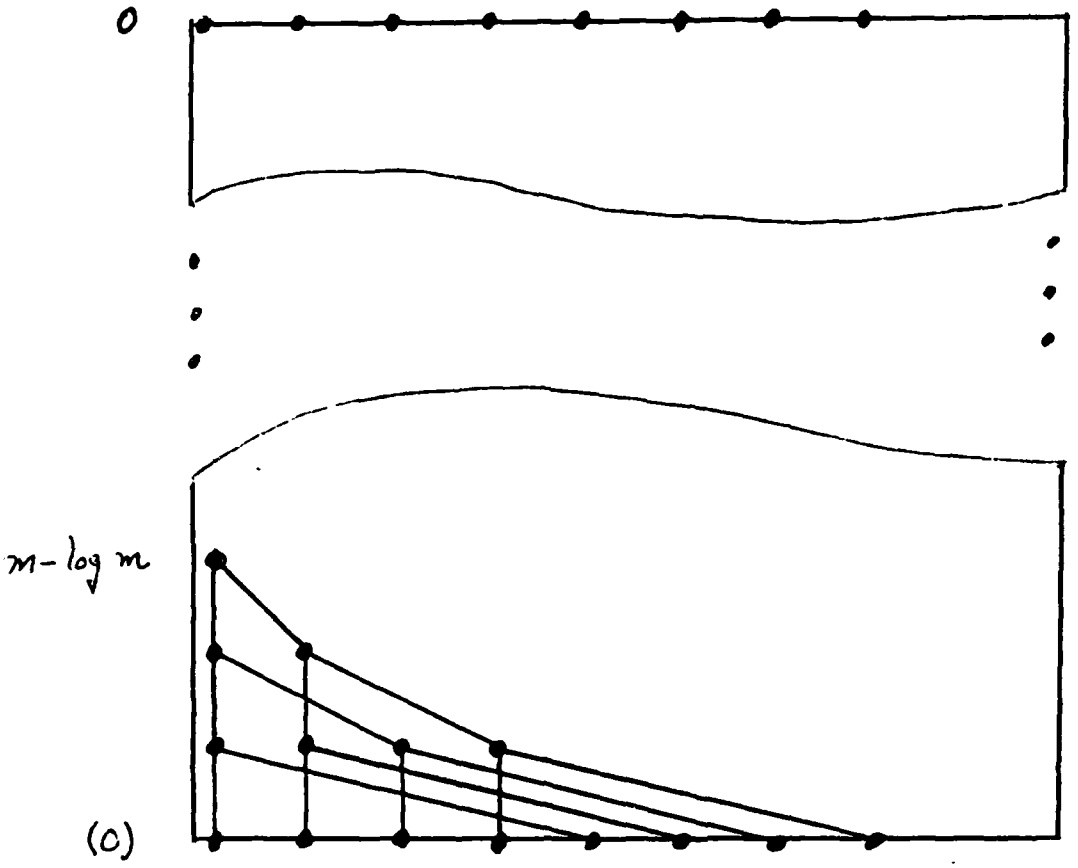


Figure 3: A logical view of the first $\log m$ levels of the embedding

where the 1 appears in the k^{th} even position from the right (using 0-based counting); call this the *signatory* 1 of the tree position. For instance, when $m = 8$, the second subtree has leaves in positions

$$\langle 7,\ *0*1*0*0?\rangle$$

of $B(9)$. We embed these $(m/2+1)$-level trees by alternating binary and unary branchings in $B(m+1)$, starting at the "roots" placed at level-0 vertices of $B(m+1)$ during Stage 1; we place a tree-vertex after each unary branching. See Fig. 4. Binary branchings generate the $*$'s in the code for the set of PWL strings, while unary branchings generate the 0's and 1's in the code. As a simple example: a binary branching from vertex

$$\langle 0,\ 000000011\rangle,$$

which holds the root of one of the subtrees planted during Stage 1, generates vertices

$$\langle 1,\ *00000011\rangle;$$

a unary branching thence generates vertices

$$\langle 2,\ *00000011\rangle,$$

where we place the level-1 vertices of the subtree; a second binary branching generates vertices

$$\langle 3,\ *0*000011\rangle;$$

a unary branching thence generates vertices

$$\langle 4,\ *0*100011\rangle,$$

where we place the level-2 vertices of the subtree; a subsequent sequence of alternating binary and unary branchings finally embeds the desired set of leaf positions in the advertised vertices of $B(m+1)$.

This stage of our embedding clearly has dilation 2. The fact that that this stage is one-to-one (though it may produce conflicts with the embedding from Stage 1) has two origins. First, we are using levels 0 through m of $B(m+1)$ for the $m+1$ levels of this stage, so the leaves of the embedded trees do not wrap around to conflict with their roots. Second, each signatory 1, whose placement identifies its respective tree, is set "on" *before* the signature bits are reached and altered by the sequence of branchings. This is ensured by the fact that we place the signatory 1 by counting from the right: the signature bits occupy the rightmost $\log m - 1$ bits

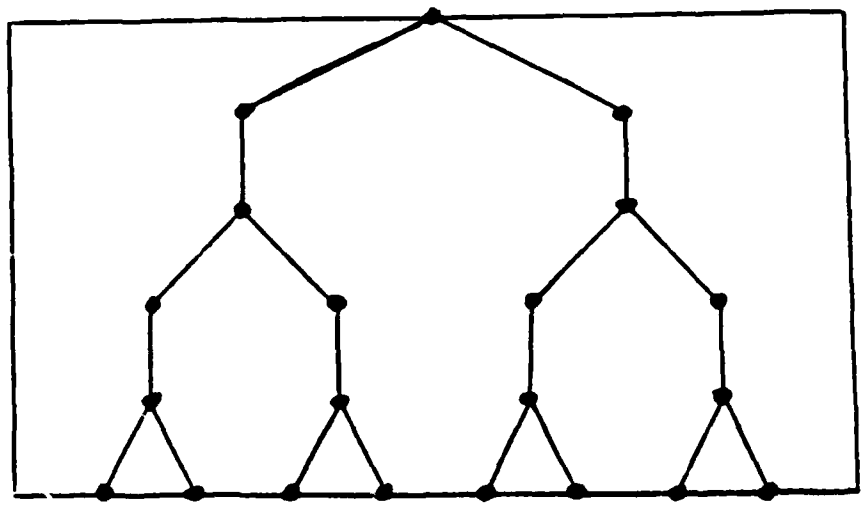


Figure 4: A logical view of the next $m/2$ levels of the embedding

of the PWL string; by the time the branchings have reached the i^{th} bit from the right, only the rightmost $(\log i)$ bits of the signature are needed to specify the next position where branching occurs. Hence, at the point when we place the signatory 1 in the i^{th} position, the odd-numbered positions to the left of the 1 are all 0, and the positions to the right of the 1 form the binary representation of i, possibly with leading 0's.

D. Stage 3: The Final $m/2$ Levels of $T(q)$

Our goal in Stage 3 is to use the $m \cdot 2^{m/2-1}$ leaves of the $m/2$ trees generated in Stage 2 as the roots of the $(m/2+1)$-level subtrees comprising the bottom $m/2$ levels of $T(q)$. Each root has a signatory 1, identifying the subtree it came from in Stage 2, and a *serial number* obtained from the odd-numbered bits of its PWL string. The signatory 1's will keep trees sired by different Stage-2 trees disjoint; the serial numbers will guard against collisions among trees that were sired by the same Stage-2 tree. The main challenge here is to achieve the embedding while the roots of all the trees reside at the same level of $B(m+1)$ (which is how Stage 2 has placed them). To accomplish this, we have the trees grow *upward*, in the direction of lower level-numbers, for varying amounts of time, before starting to grow *downward*, in the direction of higher level-numbers. While growing either upward or downward, a tree grows via alternating unary and binary branchings, *so as to preserve the serial number*; this alternation will incur dilation 2. An additional dilation of 2 is incurred while a tree grows upward: each tree begins to grows upward using only

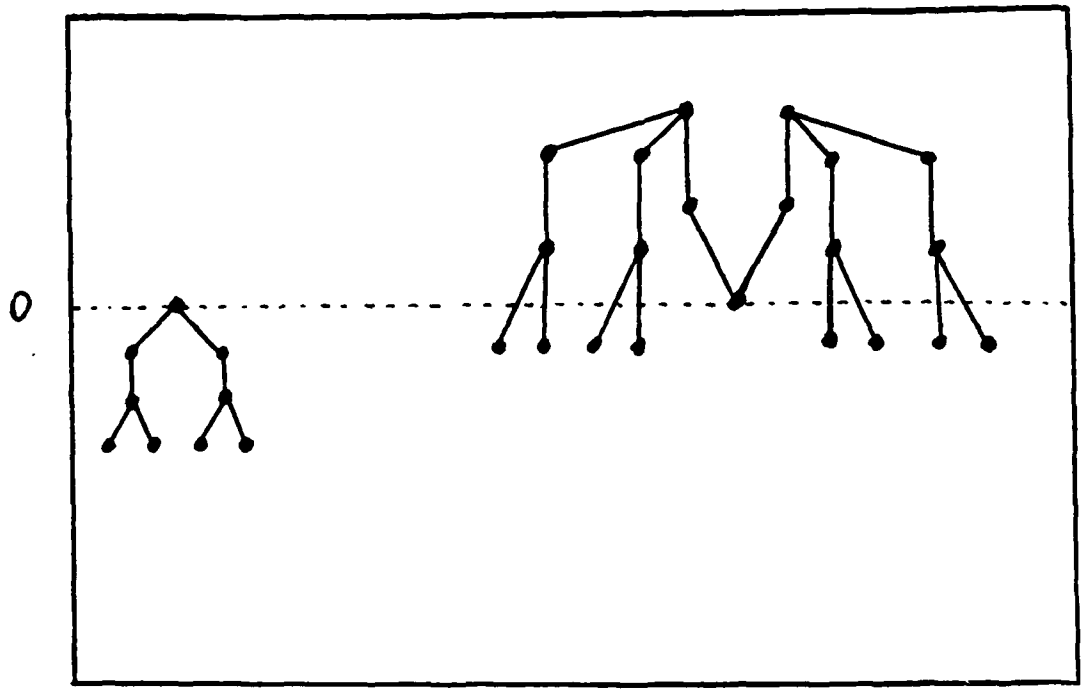


Figure 5: A logical view of the final $m/2$ levels

every fourth level of $B(m+1)$; when it "turns" from growing upward to growing downward, it uses the levels it has skipped while moving upward to regain level 0 of $B(m+1)$, at which time it grows downward using every other level of $B(m+1)$. See Fig. 5. Thus, in all, this Stage of the embedding incurs dilation 4.

All trees with the same signatory 1 (i.e., rooted at the leaves of the same Stage-2 tree) will grow in lockstep. We refer to the trees sharing a signatory 1 in the k^{th} even bit-position as the k^{th} *subtrees* of $T(q)$, $0 \leq k < m/2$. We place the vertices of the k^{th} subtrees of $T(q)$ into $B(m+1)$ as follows:

- For the 0^{th} trees, we place the 2^ℓ level-ℓ vertices of $T(q)$ at level 2ℓ of $B(m+1)$. (Thus, these trees grow downward immediately.)
- For the k^{th} trees, $k > 0$:
  - we place their unique level-0 vertex at level 0 of $B(m+1)$ (in fact this was placed during Stage 2)

- for $1 \leq \ell \leq \lfloor k/2 \rfloor$, we place their 2^ℓ level-ℓ vertices at level $m - 4\ell + 1$ of $B(m+1)$
- *if k is odd*, we place their $2^{\lceil k/2 \rceil}$ level-$(\lceil k/2 \rceil)$ vertices at level $m - 4\lceil k/2 \rceil + 3$ of $B(m+1)$
- for $\lceil k/2 \rceil + 1 \leq \ell \leq k$, we place their 2^ℓ level-ℓ vertices at level $m - 4(k - \ell) - 1$ of $B(m+1)$

Now we verify that the described mapping is one-to-one, hence an embedding. We consider separately the two potential sources of collisions.

First, we note that there can be no collisions among the $2^{m/2}$ k^{th} trees, for any k, since each of these trees has a unique serial number.

Second, we note that, for each fixed serial number, there can be no collision between the j^{th} and k^{th} trees having that serial number. This is argued most easily by considering how such trees are laid out level by level. To simplify exposition, we present only the even bit-positions of the image vertices in $B(m+1)$, since the odd bit-positions hold identical serial numbers. Note first that the top k levels of each k^{th} tree are placed in vertices of the form

$$\langle \ell,\ 0^{m/2-k} 1 *^k \rangle$$

in $B(m+1)$; hence, their membership in a k^{th} tree is announced by the leftmost $m/2 - k + 1$ even bit-positions of the PWL strings. For tree-levels $> k$, the j^{th} and k^{th} trees are distinguished as follows. Say, with no loss of generality, that $j < k$. For each $0 \leq \ell \leq m/2 - k$, the level-$(k + \ell)$ vertices of each k^{th} tree are placed at vertices

$$\langle \ell,\ *^\ell 0^{m/2-k-\ell} 10^k \rangle$$

of $B(m+1)$. By the same token, for each $0 \leq \ell \leq m/2 - j$, the level-$(j + \ell)$ vertices of each j^{th} tree are placed at vertices

$$\langle \ell,\ *^\ell 0^{m/2-j-\ell} 10^j \rangle$$

of $B(m+1)$. Since $j < k$ by hypothesis, we see that, at those levels of $B(m+1)$ where we place vertices of both trees, the k^{th} even bit-position from the right of each k^{th} tree contains a 1, while the corresponding bit-position of each j^{th} tree contains a 0.

Thus, the mapping in this stage is an embedding.

E. Resolving Collisions

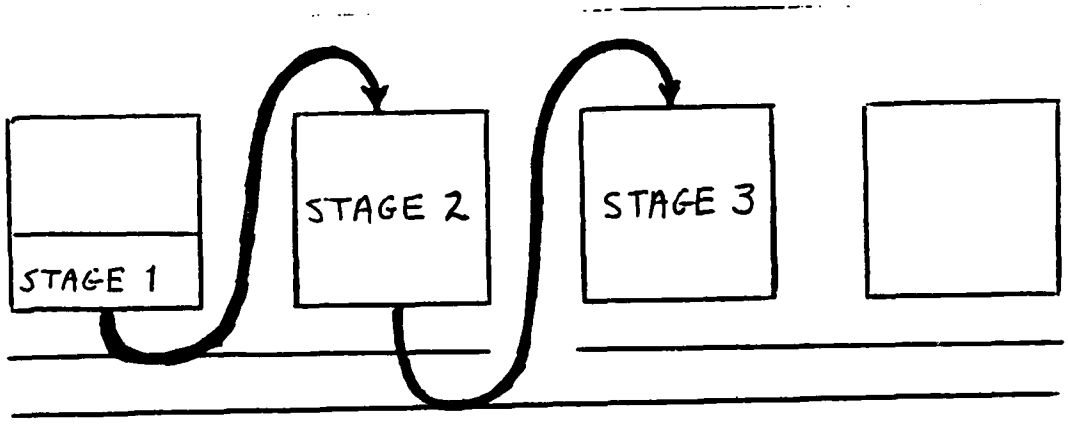


Figure 6: Replicating $B(m+1)$ to avoid collisions

We now have three subembeddings that accomplish the desired task, except for the fact that Stage i and Stage j may map different tree vertices to the same Butterfly vertex. We resolve these possible collisions as follows. Instead of performing the subembeddings in $B(m+1)$, we perform them in $B(m+3)$, placing each subembedding in a distinct copy of $B(m+1)$. We make the transition between copies of $B(m+1)$ as follows. As the Stage-1 embedding of the top of $T(q)$ reaches level $m-1$ of its copy of $B(m+1)$, we use a sequence of unary branchings in $B(m+3)$ to reach level 0 of the next copy of $B(m+1)$. We perform the Stage-2 subembedding within this second copy; this takes us to level $m-1$ of that copy, where a sequence of unary branchings in $B(m+3)$ takes us to level 0 of the third copy of $B(m+1)$. We perform the Stage-3 subembedding in this third copy. See Fig. 6. The transition from level $m-1$ of the second copy of $B(m+1)$ to level 0 of the third copy engenders dilation 4.

The embedding, hence the proof, is now complete. □

2.3. The Issue of Optimality

Theorem 1 settles for an embedding of complete binary trees in Butterfly graphs, that achieves dilation $O(1)$ and expansion $O(1)$ simultaneously. While this achieves our overall goal of optimality to within constant factors, it does leave open the possibility of those constant-factor improvements. We have been unable to determine exact dilation-expansion tradeoffs for embeddings of complete binary trees in Butterfly graphs, but we can show easily that it is impossible to optimize both cost

measures simultaneously. Thus, one cannot hope for the level of "perfection" found in, say, [GHR][6].

Proposition 2 *No embedding of $T(q)$ in $B(m+1)$ has unit dilation.*

Proof. Both complete binary trees and Butterfly graphs are bipartite graphs: one can color the vertices of either graph red and blue in such a way that every edge connects a red vertex and a blue one. For any Butterfly graph $B(r)$, on the one hand, the numbers of red and blue vertices are within r of being equal; for any complete binary tree, on the other hand, one of the sets has roughly twice as many vertices as the other. Thus, one cannot find a unit-dilation embedding of a complete binary tree in the smallest Butterfly graph that has enough vertices to hold it. □

3. UPPER BOUNDS – THEOREM 2

This section is devoted to proving Theorem 2. Since all of the relevant ideas in the proof are present in its application to specific families of graphs, we actually prove only the upper bound of Corollary 1. The reader should be able to generalize easily to arbitrary families of graphs, thereby proving Theorem 2. For the remainder of the Section, we therefore focus on the problem of embedding X-trees in Butterflies.

Our embedding of the X-tree in the Butterfly graph is indirect: First we find a unit-expansion, dilation-$O(\log\log n)$ embedding of $X(h)$ in $T(h)$. Then we compose this embedding with the expansion-$O(1)$, dilation-$O(1)$ embedding of $T(h)$ in $B(m)$ from Theorem 1, to obtain the upper bound of Theorem 2. We discuss here only the former embedding, which, in fact, embeds the X-tree $X(m)$ in the complete binary tree $T(m)$. For notational simplicity, let $n =_{\text{def}} 2^{m+1} - 1$, the number of vertices in $X(m)$. We devote this section to proving the following.

Proposition 3 *For any integer m, one can embed the X-tree $X(m)$ in the complete binary tree $T(m)$, with dilation $O(\log m) = O(\log\log n)$.*

Using the obvious fact that the n-vertex X-tree can be bisected (in the sense of statement 1 above) by removing $O(\log n)$ edges, coupled with techniques in Section 4 of [BL], the reader can easily prove the following.

[6] In [GHR] a variant of $B(m)$ with no wraparound is embedded in the Hypercube with unit dilation and optimal expansion.

Lemma 1 *For all positive integers n, k, the n-vertex X-tree has a k-color $\sqrt{2}$-bifurcator of size $S = 2k \cdot \log n$.*

Proof of Proposition 3. Our embedding uses the following auxiliary structure, which appears (in slightly different form) in [BCLR]. A *bucket tree* is a complete binary tree, each of whose level-ℓ vertices has *(bucket) capacity*

$$c \cdot \log\left(\frac{n}{2^\ell}\right)$$

for some fixed constant c to be chosen later (in Lemma 2). We embed $X(m)$ in $T(m)$ in two stages: First, we embed $X(m)$ in a bucket tree, via a many-to-one function μ that "respects" bucket capacities (always placing precisely $c \cdot \log((2^{m+1} - 1)/2^\ell)$ vertices of $X(m)$ in each level-ℓ vertex of the bucket tree) and has constant "dilation". Then we "spread" the contents of the bucket tree's buckets within $T(m)$, to achieve an embedding of $X(m)$ in $T(m)$, with the claimed dilation. Formally, the first stage of the embedding is described as follows.

Lemma 2 *Every X-tree $X(m)$ can be mapped onto a bucket tree in such a way that:*
(a) *exactly*

$$N(\ell) = 14 \log\left(\frac{2^{m+1} - 1}{2^\ell}\right) + 24$$

vertices of $X(m)$ are mapped to each level-ℓ vertex of the bucket tree, and
(b) *vertices that are adjacent in $X(m)$ are mapped to buckets that are at most distance 5 apart in the bucket tree.*

> The constants in the expression for $N(\ell)$ can be reduced by increasing the constant 5 in part (b) of the Lemma (say, to 10). We suffer the larger constants in order to simplify the technical development in the proof. The interested reader can easily mimic our development with other constants.

Proof. The basic idea is to recursively bisect $X(m)$, using a 5-color $\sqrt{2}$-bifurcator (the uses of the colors will become clear momentarily), placing successively smaller sets of $\sqrt{2}$-bifurcator vertices in lower-level buckets of the bucket tree. We also place other vertices in the buckets, in order to ensure the desired "dilation" and in order to ensure that all buckets are filled to capacity. The formal description of the mapping will require two iterations. First, we present a mapping procedure that establishes the sufficiency of the quantities $N(\ell)$ as bucket capacities. Then we refine the initial mapping to complete the proof.

We simplify our description of this technically cumbersome procedure in two ways. First, we describe in detail what the procedure would look like if we were using *3-color bifurcators* rather than 5-color bifurcators; the reader should be able to extrapolate from oor description to arbitrray numbers of colors. Second, we establish the following notation.

- We denote by B_λ, where λ denotes the null string (i.e., the string of length 0) over the alphabet $\{1,2\}$, the bucket at the root of the bucket tree.
- In general, letting x denote any string over the alphabet $\{1,2\}$, we denote by B_{x1} and B_{x2} the buckets at the children of the vertex of the bucket tree having bucket B_x; for example, B_1 and B_2 denote the buckets at the children of the root vertex of the bucket tree, B_{11} and B_{12} denote the buckets at the left grandchildren of the root vertex, B_{21} and B_{22} denote the buckets at the right grandchildren of the root vertex, and so on.

Algorithm Bucket: Mapping $X(m)$ into a bucket tree

Step 1. Initial coloring and bisection.

1.a. Initialize every vertex of $X(m)$ to color A.

1.b. Associate[7] the graph $X(m)$ with the root of the bucket tree.

1.c. Bisect $X(m)$, to obtain subgraphs X_1 and X_2, and place the $\sqrt{2}$-bifurcator vertices in bucket B_λ.

1.d. Recolor every A-colored vertex of $X(m)$ that is adjacent to a vertex in bucket B_λ with color 0.

1.e. Associate X_i ($i \in \{1,2\}$) with the child of the root vertex of the bucket tree holding bucket B_i.

Step 2. Second-level bisection.

2.a. Use a 2-color $\sqrt{2}$-bifurcator for each X_i, to create subgraphs X_{i1} and X_{i2}.

2.b. Place the $\sqrt{2}$-bifurcator vertices for each X_i in the corresponding bucket B_i of the bucket tree.

2.c. Recolor every A-colored vertex of $X(m)$ that is adjacent to a vertex in bucket B_i with color 1.

[7] The "associations" here are intended to make it easier for the reader to follow our description of the mapping.

2.d. For each X_i, associate each subgraph X_{ij} with the $\sqrt{2}$-bifurcator-tree vertex associated with bucket B_{ij}.

Step 3. Third-level bisection.

3.a. Use a 3-color $\sqrt{2}$-bifurcator for each X_{ij}, to create subgraphs X_{ij1} and X_{ij2}.

3.b. Place the $\sqrt{2}$-bifurcator vertices for each X_{ij} in the corresponding bucket B_{ij} of the bucket tree.

3.c. Recolor every A-colored vertex of $X(m)$ that is adjacent to a vertex in bucket B_{ij} with color 0.

3.d. For each X_{ij}, associate each subgraph X_{ijk} with the $\sqrt{2}$-bifurcator-tree vertex associated with bucket B_{ijk}.

Step s. ($4 \leq s \leq m$) All remaining bisections.

s.a. For each subgraph X_y ($y \in \{1,2\}^*$) of $X(m)$ created in Step $s-1$, place every vertex of color $s \pmod 2$ in the associated bucket B_y.

s.b. Use a 3-color $\sqrt{2}$-bifurcator for each X_y, to create subgraphs X_{y1} and X_{y2}.

s.c. Place the $\sqrt{2}$-bifurcator vertices for each X_y in the corresponding bucket B_y of the bucket tree.

s.d. Recolor every A-colored vertex of $X(m)$ that is adjacent to a vertex in bucket B_y with color $length(y) \pmod 2$.

s.e. For each X_y, associate each subgraph X_{yi} with the $\sqrt{2}$-bifurcator-tree vertex associated with bucket B_{yi}.

We now analyze *5-color analogue of the described mapping*, to show that it satisfies the demands of Lemma 2, with the requirement of "exactly" $N(\ell)$ vertices per level-ℓ bucket replaced by "no more than" $N(\ell)$ vertices per level-ℓ bucket, i.e., to show that our bucket capacities are big enough. Since the "dilation" condition (b) is transparently enforced when certain colored vertices are automatically placed in buckets (in Step s.a), it will suffice to establish that the populations of the buckets are as indicated in the modified condition (a). This follows by the following recurrence, wherein $N(k)$ denotes the number of vertices of $X(m)$ that get mapped into a bucket at level $k-1$ of the bucket tree.

$$\begin{aligned} N(k) &\leq \left\lceil \frac{5}{32} N(k-5) \right\rceil + 6 \log\left(\frac{n}{2^k}\right) \\ &\leq \frac{3}{16} N(k-5) + 10 \log\left(\frac{n}{2^k}\right) \end{aligned}$$

with initial conditions

- $N(1) \leq 2 \log n$
- $N(2) \leq 4 \log \left(\frac{n}{2}\right)$
- $N(3) \leq 6 \log \left(\frac{n}{4}\right)$
- $N(4) \leq 8 \log \left(\frac{n}{8}\right)$
- $N(5) \leq 10 \log \left(\frac{n}{16}\right)$

The initial conditions reflect the sizes of the appropriately colored $\sqrt{2}$-bifurcators of $X(m)$: At each level ℓ, $1 \leq \ell \leq 4$, one uses an ℓ-colored $\sqrt{2}$-bifurcator, followed by a 5-color $\sqrt{2}$-bifurcator at all subsequent levels. At levels $s > 2$, the buckets contain not only $\sqrt{2}$-bifurcator vertices, which account for the term

$$10 \log \left(\frac{n}{2^k}\right)$$

in the general recurrence; they contain also the vertices of $X(m)$ that are placed to satisfy the "dilation" requirements. The latter vertices comprise all neighbors of the $N(k-5)$ occupants of the distance-4 ancestor bucket that have not yet been placed in any other bucket. Since vertices of $X(m)$ can have no more than five neighbors, and since our 5-color bisections allocate these neighbors equally among the descendants of a given bucket, these "dilation"-generated vertices can be no more than

$$\left\lceil \frac{5}{32} N(k-5) \right\rceil \leq \frac{3}{16} N(k-5)$$

in number. These two sources, the $\sqrt{2}$-bifurcators and their neighbors, account for the occupants of the buckets and for the recurrence counting them. To complete the proof of the modified Lemma, one now shows by standard techniques that the indicated recurrence, with the indicated initial conditions, has the solution

$$N(k) \leq 14 \log \left(\frac{n}{2^k}\right) + 24.$$

Finally, we turn to the original form of the Lemma. This follows from the modified form, upon refining the Algorithm by adding the following substeps at the indicated points.

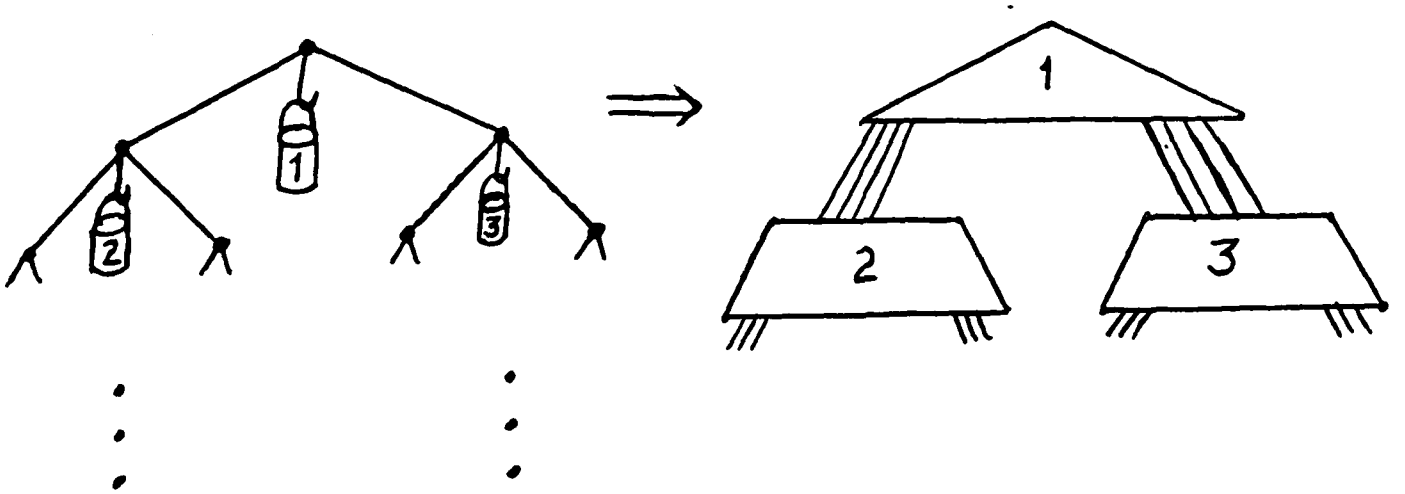


Figure 7: Unloading the buckets

> At the end of each step of the Algorithm, when we have finished filling a bucket B_x ($x \in \{1,2\}^*$) with vertices obtained from a recent bisection or from our desire to maintain small "dilation", we check the population of the bucket against the ceiling population $N(\ell)$, where $\ell = \mathit{length}(x)$. If the bucket contains fewer than $N(\ell)$, vertices, then we add enough new vertices to it from the remaining associated subgraph to fill it to capacity.

This last step ensures that all buckets at level ℓ of the bucket tree contain exactly $N(\ell)$ vertices. □

Our final task is to refine the "dilation"-5 mapping of Lemma 2 to a bona fide embedding of $X(m)$ in $T(m)$, having dilation $O(\log\log n)$. We proceed inductively, emptying buckets into $T(m)$ in such a way that each tree vertex is assigned a unique X-tree vertex. In general, we denote by T_x the smallest subtree of $T(m)$ that is rooted at level $\mathit{length}(x)$ of $T(m)$ and that contains the contents of bucket B_x. (In general, the contents of B_x will occupy only the last few levels of T_x.) See Fig. 7.

- Place the $\log n$ elements of bucket B_λ in the topmost copy of $T(\log\log n)$ in $T(m)$, in any way.
- Consider the subtrees of T_λ rooted at level 1 of $T(m)$. Place the contents of bucket B_1 in the (roughly) $\log\log n$ levels of the leftmore of these two subtrees,

starting immediately after the leaves of T_λ. Place the contents of bucket B_2 analogously, using the rightmore of these two subtrees, starting immediately after the leaves of T_λ. We have thus implicitly defined the subtrees T_1 and T_2.

Note that by this point, we are using enough of the top levels of $T(m)$ that we need use only one more level in order to place the contents of the next level of buckets. The importance of this fact is that it guarantees that all of the subtrees T_x will have height $O(\log\log n)$. (Namely, T_λ, T_1, and T_2 have the desired height, and all subsequent trees will result from adding one level of leaves to a tree whose root is one level lower in $T(m)$ than was its father's root.)

- Proceeding inductively, assume that we have filled subtrees T_x of $T(m)$ with bucket contents, for strings $x \in \{1,2\}^*$ of length $\leq \ell$. We now consider the subtrees of $T(m)$ rooted at level $\ell+1$; each subtree T_x rooted at level ℓ thus spawns two children. We order these $2^{\ell+1}$ subtrees from left to right, according to the lexicographic order on the subscript-strings x. We then place the contents of the bucket B_{x1} in the leaves of the leftmore of the children of T_x, beginning where the contents of bucket B_x left off. Analogously, we place the contents of the bucket B_{x2} in the leaves of the rightmore of the children of T_x, beginning where the contents of bucket B_x left off.

The described procedure clearly produces an embedding of $X(m)$ in $T(m)$, since each vertex of $X(m)$ is assigned to a unique tree vertex. Additionally, the embedding has unit expansion since no tree vertices are passed over in the assignment process and since all buckets at each level ℓ have the same population $N(\ell)$ (so all subtrees T_x are isomorphic). Finally, the procedure's method of spreading bucket contents throughout $T(m)$ produces an embedding with the desired dilation, namely, $O(\log\log n)$. Specifically, by always spreading the contents of buckets B_{x1} and B_{x2} in the leaves of the left and right subtrees of the depth-$O(\log\log n)$ subtree that contains the contents of bucket B_x, the procedure guarantees that the least common ancestor, in $T(m)$, of the set comprising the contents of any bucket plus the vertices in buckets at most five buckets up (which will lie in adjacent levels $k, k+1, k+2, k+3, k+4, k+5$ of the bucket tree) are always within a subtree of height $O(\log\log n)$ of $T(m)$. Thus, we have produced the desired embedding, thereby proving Proposition 2, hence Theorem 2. □

4. LOWER BOUNDS – THEOREM 3

We demonstrate the near-optimality (to within constant factors) of the embeddings of Section 3 – in fact, true optimality for X-trees – by proving the lower bound of

Theorem 3. In contrast with Theorem 2, Theorem 3 is most easily proved in its full generality.

Assume henceforth that we are given a planar graph G, a planar embedding ϵ of G, and a minimum-dilation embedding μ of G in $B(p)$; let μ have dilation δ.

We begin by noting that we can simplify our quest somewhat. Specifically, since we aim only for bounds that hold up to constant factors, we lose no generality by assuming henceforth that (in the embedding ϵ) the exterior face of G is a simple cycle:

Lemma 3 *One can add edges to the graph G within the embedding ϵ in such a way that*

- *the resulting embedding ϵ' is a planar embedding of the resulting graph G'*
- *in the embedding ϵ', the exterior face of G' is a simple cycle*
- $\Sigma(G') = O(\Sigma(G))$
- $\Phi_{\epsilon'}(G') = \max(3, \Phi_{\epsilon}(G))$.

Proof Sketch. If the exterior face of G is not a simple cycle, it is because of cut-edges and/or pinch-vertices. We take each cut-edge in turn and create a triangle containing it as an edge; then we repeat the process with any remaining cut-edge. When no more cut-edges exist, we eliminate each pinch-vertex in turn by creating a triangle that includes the pinch-vertex as a vertex. Since each added edge creates a triangle and spans only two edges of G, the claims about $\Phi(G')$ and $\Sigma(G')$ are immediate. □

A consequence of Lemma 3 is that we may henceforth assume that every edge of G resides in some interior face (in the embedding ϵ).

We turn now to the quantitative consequences of Lemma 3.

A set of faces of G is *connected* in the embedding ϵ just when their corresponding vertices are connected in the graph $\Gamma(G;\epsilon)$ whose vertices are the faces of G and whose edges connect a pair of face-vertices just when the faces share a vertex. A set S of vertices of G is *face-connected* (in ϵ) if the set of *interior faces* of G that contain one or more of the vertices of S is connected.

Let A be a connected component of the graph G remaining after removing a set S of vertices from G. The *S-boundary of A* is the set of vertices of A that are adjacent (in G) to vertices of S.

Lemma 4 *If one removes a face-connected set of vertices S from the graph G, then the S-boundary of every resulting maximal connected component of G is face-connected.*

Proof. Consider a maximal connected component A remaining after removing S from G. Assume for contradiction that the set of S-boundary vertices of A is not face-connected. There must then be at least two distinct maximal connected components, call them F_1 and F_2, of interior faces that contain boundary vertices (so $F_1 \cup F_2$ is not connected). Let f_i, $i = 1, 2$, be an interior face in component F_i, and let b_i be a boundary vertex in face f_i. Since each edge of G lies in an interior face, we can choose each f_i to contain a vertex of S as well as a boundary vertex.

Fact 1 *There is a connected set I of interior faces, none of which contains a boundary vertex, such that I separates f_1 from f_2.*

Verification. It is not possible for both F_1 to encircle f_2 and F_2 to encircle f_1, since then F_1 and F_2 would intersect (so f_1 and f_2 would be connected by interior faces containing boundary vertices). Without loss of generality, say that F_1 does not encircle f_2.

Let J be the set of interior faces that do not contain boundary vertices and that are incident to the outer boundary of F_1 (so that f_2 is on the outside). By definition, the set J separates f_1 from f_2. If J is connected, then it is the desired set I. If J is not connected, then adding the exterior face of G to J yields a connected set J'. Moreover, f_2 must lie in one of the simply connected regions J'' of J'. Deleting the exterior face from J'' then yields the desired set I; see Fig. 8.

Fact 2 *I contains a vertex of A and a vertex of S.*

Verification. I separates f_1 from f_2, yet: f_1 and f_2 both contain vertices of both A and S; both A and S are face-connected in G.

Since I contains vertices of A and S and is connected, and since S separates the connected set A from the rest of G, the set I must contain at least one face that contains *both* a vertex from A and a vertex from S. Such a face must also contain a vertex of the S-boundary of A, contradicting Fact 1. Lemma 4 follows. □

A set of vertices S of a graph K is *d-quasi-connected*, d a positive integer, if for every two vertices u, w of S, there exists a chain of vertices

$$u = v_0, v_1, v_2, \ldots, v_k = w,$$

of S, where consecutive vertices v_i, v_{i+1} are distance $\leq d$ apart in K.

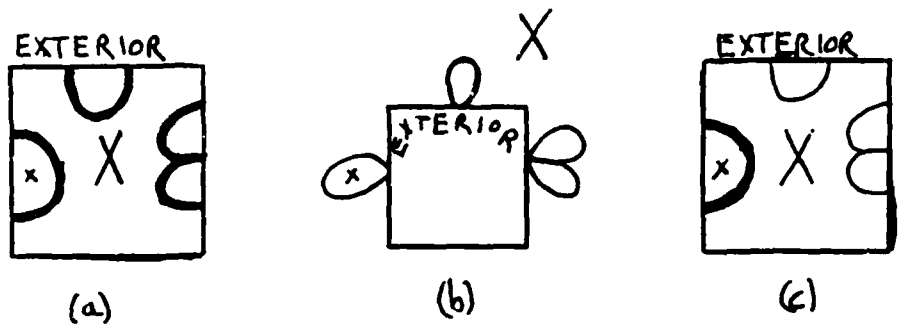


Figure 8: (*a*) The embedding ϵ, with the set J outlined boldly; "x" marks f_2, and "X" marks F_2 with the holes filled in. (*b*) The set $J' = J \cup$ (outer face). (*c*) The embedding ϵ, as in (a), with the set J'' outlined boldly.

Lemma 5 The Fundamental Lemma for Butterfly-Like Graphs
Say that there is a subgraph K of G and a constant c such that

- *K is $\Phi(G)$-quasi-connected*
- *the image of K under the embedding μ lies within $c\Phi(G)\delta$ consecutive levels of $B(p)$.*

Then $\delta \geq \alpha(c)\frac{\log|K|}{\Phi(G)}$, where $\alpha(c)$ is a constant depending only on c.

Proof. Say that the image of H under μ lies entirely in levels[8]

$$l+1, l+2, \cdots, l+c\Phi(G)\delta$$

of $B(p)$. Let u and v be arbitrary vertices of H which are connected by a path of at most $\Phi(G)$ vertices in G. The image of this path in $B(p)$ must lie totally within levels

$$l-\Phi(G)\delta+1, \cdots, l+(c+1)\Phi(G)\delta$$

of $B(p)$, since the embedding μ has dilation δ. See Fig. 9. Since K is $\Phi(G)$-quasi-connected, this means that the PWL strings of *all* images of vertices of K can differ

[8] All addition is modulo p.

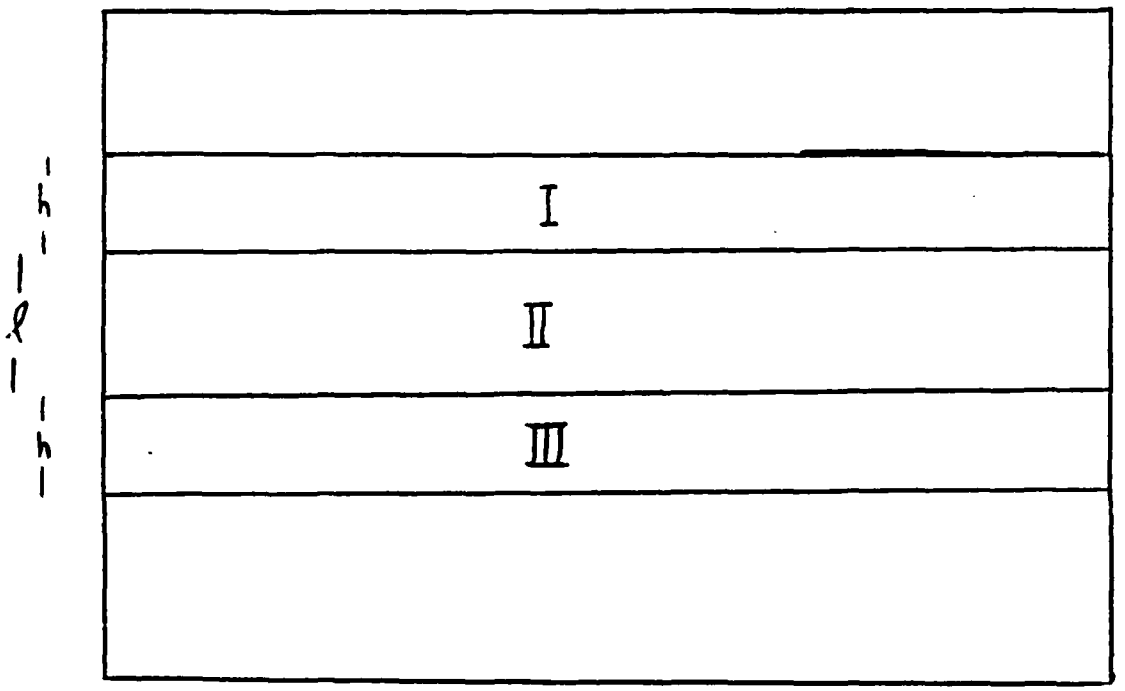


Figure 9: Illustrating the Fundamental Lemma, with $\ell = c\Phi(G)\delta$ and $h = \Phi(G)\delta$: Vertices of K reside in region II; length-$\Phi(G)$ paths between vertices of K cannot extend beyond regions I or III.

only in some set of at most $((c+2)\Phi(G)+1)\delta$ bit positions. It follows that K can contain no more than $c\Phi(G)\delta 2^{((c+2)\Phi(G)+1)\delta}$ vertices, i.e., $c\Phi(G)\delta$ levels of $B(p)$, with at most $2^{((c+2)\Phi(G)+1)\delta}$ vertices per level. In other words,

$$c\Phi(G)\delta 2^{((c+2)\Phi(G)+1)\delta} \geq |K|,$$

whence the result. □

We now complete the proof of Theorem 3, beginning with two simple lemmas.

Lemma 6 *Any face-connected set of vertices of G is $\Phi(G)$-quasi-connected.*

Proof Sketch. Any vertex in an f-vertex face is distance $\leq f/2$ from any neighboring face. □

Lemma 7 *Let C be a set of vertices of the graph G whose removal partitions G into connected components all of size $\leq |G|/2$. Then C is a 1/3-2/3 separator of G.*

Proof Sketch. Remove C from G, order the resulting connected components by size into decreasing order, and lump the components into two piles as follows.

- Place the largest component into the left pile.
- Place as few of the largest remaining components in the right pile as possible until the right pile is bigger than the left.
- Now alternate piles, adding as few of the largest remaining piles as possible to the smaller pile until the smaller first becomes bigger than the larger pile.

Clearly, when one has completed the two piles, the larger cannot be bigger than the smaller by more than the size of the third largest component, i.e., by more than $|G|/3$ vertices. It follows that each pile must contain at least $|G|/3$ vertices, whence the claim. □

Theorem 3 will now follow from the next Lemma.

Lemma 8 *The embedding μ must have dilation $\delta \geq (const)\left(\frac{\log \Sigma(G)}{\Phi(G)}\right)$.*

Proof. Partition $B(p)$ into *bands*, each band β_i being a sequence of $d_i\delta$ consecutive levels, $2\Phi(G) \leq d_i < 4\Phi(G)$, where the constants d_i may be chosen in any way that achieves a partition. Let $\kappa(v)$, the *color* of vertex v of G, be the index i of the band β_i in which $\mu(v)$ resides.

We perform a modified breadth-first search of G, to find a $\Phi(G)$-quasi-connected component of size $\geq \Sigma(G)$, all of whose vertices have images in a single band of $B(p)$, hence the same color. By Lemma 5, the existence of such a component will yield the lower bound on δ.

The breadth-first search proceeds as follows. We select an arbitrary vertex v_0 of G and form V_0, the maximal connected component of G that contains v_0 and that consists entirely of vertices with color $\kappa(v_0)$. Since V_0 is connected, removing its vertices partitions G into connected components; let C_0 be the largest of these. Lemmas 4 and 6 assure us that the V_0-boundary, B_0, of the component C_0 is $\Phi(G)$-quasi-connected. It follows that

Fact 3 *All vertices of B_0 have the same color.*

Verification. Since each $v \in B_0$ is adjacent to a vertex of V_0, we must have $\kappa(v) \in \{\kappa(v_0) - 1, \kappa(v_0) + 1\}$. Moreover, B_0 cannot contain vertices of both colors: Two such vertices would be separated by the band $\beta_{\kappa(v_0)}$, contradicting the fact that B_0 is $\Phi(G)$-quasi-connected.

Next, form V_1, the maximal *monochromatic* subgraph of G that contains both B_0 and all connected components of G that intersect B_0; obviously, V_1 is $\Phi(G)$-quasi-connected, so removing it partitions G into some number of connected components. Let C_1 be the largest of these, and let B_1 be the V_1-boundary of C_1. As with B_0, one shows that B_1 is $\Phi(G)$-quasi-connected and monochromatic.

We continue in this fashion, constructing, in turn, for $i = 2, 3, \ldots$, the following subgraphs of G, with the indicated properties:

- V_i: the ($\Phi(G)$-quasi-connected) maximal monochromatic subgraph of G that contains both B_{i-1} and all connected components of G that intersect B_{i-1}
- C_i: the largest connected component of G remaining when one removes V_i from G
- B_i: the ($\Phi(G)$-quasi-connected, monochromatic) V_i-boundary of C_i

One continues this construction until some subgraph V_i contains at least $\Sigma(G)$ vertices. We now show that this point must occur.

Fact 4 *For some i, $|V_i| \geq \Sigma(G)$.*

Verification. Note that at each point in our construction, V_i is whittled out of the largest component C_{i-1} of G remaining after removal of V_{i-1} from G. Moreover, V_i disconnects the vertices of $C_{i-1} - V_i$ from the remainder of G, as one can verify easily by induction on i. At some point, therefore, the whittling process must reduce the size of the then-current largest component C_m so that $|C_m| \leq |G|/2$. By Lemma 7, the then-current V_n is a 1/3-2/3 separator of G, hence must contain at least $\Sigma(G)$ vertices.

The preceding development gives us a set of vertices, of size $\geq \Sigma(G)$, whose images reside in a single band of d_i levels of $B(p)$. By Lemma 5, Theorem 3 follows. □

5. THE COROLLARIES: X-TREES AND MESHES

Corollaries 1 and 2 now follow from the following Lemmas.

Lemma 9 $|\mathrm{HR}|$ $\Sigma(X(h)) = \Omega(h) = \Omega(\log |X(h)|)$, *and* $\Phi(X(h)) = 5$ *(under the natural embedding).*

Lemma 10 (e.g., $|\mathrm{HR}|$) $\Sigma(M(s)) = \Omega(s) = \Omega(\sqrt{|M(s)|})$, *and* $\Phi(M(s)) = 4$ *(under the natural embedding).*

6. CONCLUDING REMARKS

We close with some remarks about extensions to the research described here.

The lower bound of Theorem 3 cannot be improved in general, as one can see from considering homeomorphs of the mesh.

Our lower bound for the mesh extends also to higher-dimensional meshes and to pyramid graphs; thus, these are examples of other popular networks that embed efficiently in the Hypercube, but not in butterfly-like machines.

The lower bound of Theorem 3, which deals explicitly only with embeddings in the Butterfly, extends to embeddings in the mesh of trees, Cube-Connected-Cycles, Benes network, and similar levelled networks.

We do not yet have an analogue of Theorem 3 for embeddings in the shuffle-exchange and deBruin graphs[9]. However, using rather complicated arguments, we can prove that any expansion-$O(1)$ embedding of the n-vertex X-tree or the n-vertex mesh in these host graphs requires dilation $\Omega(\log \log n)$. Since a complete binary tree is a spanning tree of the deBruijn graph, the proof technique of Section 3 shows that this lower bound for the X-tree is optimal. We suspect that the lower bound for the mesh can be improved.

In order to justify dilation fully as the central measure of concern in network embeddings, it would be nice to strengthen the results of Section 3 to show that the Butterfly can simulate any graph having a $\sqrt{2}$-bifurcator of size $S = \Omega(\log n)$ with delay $O(\log S)$. We believe this to be possible using the arguments of Section 2, but we have not worked through the details.

Lastly, it should be noted that our lower bounds do *not* mean that a Butterfly cannot efficiently simulate a mesh or X-tree efficiently over a large span of time. For example, a Butterfly can simulate $\log n$ steps of a mesh of a constant fraction smaller size within $O(\log n \log \log n)$ steps, and possibly within $O(\log n)$ steps. Similar improvements in *amortized* simulation times are also possible for the X-tree, and we are currently studying how good such amortized simulations can be in general.

ACKNOWLEDGMENTS: The authors wish to thank Dave Barrington and Les Valiant for helpful conversations.

7. REFERENCES

[Ag] A. Aggarwal (1984): A comparative study of X-tree, pyramid, and related machines. *25th IEEE Symp. on Foundations of Computer Science*, 89-99.

[ABR] F. Annexstein, M. Baumslag, A.L. Rosenberg (1987): Group-action graphs and parallel architectures. Tech. Rpt., Univ. of Massachusetts; submitted for publication.

[Be] V.E. Benes (1964): Optimal rearrangeable multistage connecting networks. *Bell Syst. Tech. J. 43*, 1641-1656.

[BCLR] S.N. Bhatt, F.R.K. Chung, F.T. Leighton, A.L. Rosenberg (1986): Optimal simulations of tree machines. *27th IEEE Symp. on Foundations of Computer Science*, 274-282.

[9] These are "essentially" the same graph, since each can be embedded in the other with unit expansion and dilation 2.

[BI] S.N. Bhatt and I. Ipsen (1985): Embedding trees in the hypercube. Yale Univ. Rpt. RR-443.

[BL] S.N. Bhatt and F.T. Leighton (1984): A framework for solving VLSI graph layout problems. *J. Comp. Syst. Sci. 28*, 300-343.

[DP] A.M. Despain and D.A. Patterson (1978): X-tree – a tree structured multiprocessor architecture. *5th Symp. on Computer Architecture*, 144-151.

[Ga] D. Gannon (1980): On pipelining a mesh-connected multiprocessor for finite element problems by nested dissection. *Intl. Conf. on Parallel Processing.*

[GHR] D.S. Greenberg, L.S. Heath, A.L. Rosenberg (1987): Optimal embeddings of the FFT graph in the Hypercube. Typescript, Univ. of Massachusetts; submitted for publication.

[HR] J.-W. Hong and A.L. Rosenberg (1982): Graphs that are almost binary trees. *SIAM J. Comput. 11*, 227-242.

[HZ] E. Horowitz and A. Zorat (1981): The binary tree as an interconnection network: applications to multiprocessor systems and VLSI. *IEEE Trans. Comp., C-30*, 247-253.

[Le] F.T. Leighton (1984): Parallel computation using meshes of trees. *1983 Workshop on Graph-Theoretic Concepts in Computer Science*, Trauner Verlag, Linz, pp. 200-218.

[PV] F.P. Preparata and J.E. Vuillemin (1981): The cube-connected cycles: a versatile network for parallel computation. *C. ACM 24*, 300-309.

[Ro] A.L. Rosenberg (1981): Issues in the study of graph embeddings. In *Graph-Theoretic Concepts in Computer Science: Proceedings of the International Workshop WG80*, Bad Honnef, Germany (H. Noltemeier, ed.) *Lecture Notes in Computer Science 100*, Springer-Verlag, NY, 150-176.

VLSI Memo No. 88-468
August 1988

MESSAGE-DRIVEN PROCESSOR ARCHITECTURE

William Dally, Andrew Chien, Stuart Fiske, Waldemar Horwat, John Keen, Peter Nuth, Jerry Larivee, and Brian Totty

Abstract

The Message Driven Processor is a node of a large-scale multiprocessor being developed by the Concurrent VLSI Architecture Group. It is intended to support fine-grained, message passing, parallel computation. It contains several novel architectural features, such as a low-latency network interface, extensive type-checking hardware, and on-chip memory that can be used as an associative lookup table.

This document is a programmer's guide to the MDP. It describes the processor's register architecture, instruction set, and the data types supported by the processor. It also details the MDP's message sending and exception handling facilities.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

Massachusetts Institute of Technology | **Microsystems Research Center** | Cambridge Massachusetts 02139 | Room 39-321 Telephone (617) 253-8138

VLSI Memo No. 88-450
June 1988

OBJECT-ORIENTED CONCURRENT PROGRAMMING IN CST

William J. Dally and Andrew A. Chien

Abstract

CST is a programming language based on Smalltalk-80 that supports concurrency using locks, asynchronous messages, and distributed objects. Distributed objects have their state distributed across many nodes of a machine, but are referred to by a single name. Distributed objects are capable of processing many messages simultaneously and can be used to efficiently connect together large collections of objects. They can be used to construct a number of useful abstractions for concurrency. This paper describes the CST language, gives examples of its use, and discusses an initial implementation.

VLSI Memo No. 88-470
August 1988

MICRO-OPTIMIZATION OF FLOATING-POINT OPERATIONS

William J. Dally

Abstract

This paper describes micro-optimization, a technique for reducing the operation count and time required to perform floating-point calculations. Micro optimization involves breaking floating-point operations into their constituent micro-operations and optimizing the resulting code. Exposing micro-operations to the compiler creates many opportunities for optimization. Redundant normalization operations can be eliminated or combined. Also, scheduling micro-operations separately results allows dependent operations to be partially overlapped. A prototype expression compiler has been written to evaluate a number of micro-optimizations. On a set of benchmark expressions operation count is reduced by 33% and execution time is reduced by 40%.

Microsystems Research Center
Room 39-321

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

Telephone (617) 253-8138

Acknowledgements

This work was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-80-C-0622, N00014-87-K-0825 and N00014-85-K-0124 and in part by a National Science Foundation Presidential Young Investigators Award with matching funds from General Electric Corporation and IBM Corporation.

Author Information

Dally: Department of Electrical Engineering and Computer Science, Artificial Intelligence Laboratory, MIT, Room NE43-417, Cambridge, MA 02139, (617) 253-6043.

Micro-Optimization of Floating-Point Operations[1]

William J. Dally
Artificial Intelligence Laboratory and
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

Abstract

This paper describes micro-optimization, a technique for reducing the operation count and time required to perform floating-point calculations. Micro optimization involves breaking floating-point operations into their constituent micro-operations and optimizing the resulting code. Exposing micro-operations to the compiler creates many opportunities for optimization. Redundant normalization operations can be eliminated or combined. Also, scheduling micro-operations separately results allows dependent operations to be partially overlapped. A prototype expression compiler has been written to evaluate a number of micro-optimizations. On a set of benchmark expressions operation count is reduced by 33 % and execution time is reduced by 40 %.

1 Introduction

Many unneeded operations are performed during the evaluation of floating point expressions because existing compilers and floating point units consider these operations to be atomic. By decomposing floating point operations into their constituent integer micro-operations, many opportunities for optimization are exposed. Redundant shift operations may be eliminated, parts of the computation may be done with a block exponent, common subexpressions in the mantissa or exponent calculation are exposed, and additional flexibility in scheduling operations is possible.

This paper describes methods for micro-optimizing floating point expressions. Each operation in the expression is decomposed into its primitive integer micro-operations. For example a floating point add is decomposed into an exponent subtract, mantissa alignment, mantissa add, leading zero's count, exponent adjust, and mantissa normalization. Optimizations are performed on the resulting micro-operations. For example, a normalizing left shift from one FP add may be combined with the aligning right shift of a subsequent FP add resulting in a single shift. The entire expression is scheduled as a unit resulting in better hardware utilization.

On a set of benchmark expressions, micro-optimization reduces operation count by 33 % and execution time by 40 % compared to conventional floating point execution with identical

[1]The research described in this paper was supported in part by the Defense Advanced Research Projects Agency under contracts N00014-80-C-0622 and N00014-85-K-0124 and in part by a National Science Foundation Presidential Young Investigator Award with matching funds from General Electric Corporation and IBM Corporation.

Submitted to ASPLOS '88

function unit performance and register bandwidth.

To fully exploit micro-optimization, a micro floating point unit (μFPU) is required. The instruction set of a μFPU consists of the micro-operations required for floating point arithmetic (e.g., alignment shifts that maintain guard, round, and sticky bits). These operations are performed out of a set of mantissa and exponent registers. By providing the appropriate primitive operations, no comprimises are made in terms of accuracy, rounding, adherence to standards, and performance.

This work is motivated by recent progress on RISC [8] and VLIW [3] architectures. RISC machines eliminate the complex addressing modes found in CISC machines [9]. Address calculations are performed using integer arithmetic instructions rather than by microcode or special hardware. Exposing these calculations to the compiler often improves performance. Micro optimization applies this technique to floating point operations. As with address calculations, breaking these operations into their primitive components has the disadvantage of decreasing code density and increasing instruction bandwidth.

Micro-optimization borrows from VLIW technology, in that several micro-operations may be performed simultaneously. Also, some of the optimizations described here schedule code across basic blocks. However, the technique used is different from trace scheduling.

The idea of using a compiler to optimize a function normally considered a primitive arithmetic operation has been applied to integer multiplication by a constant [5].

The next section illustrates the basic concepts of micro-optimization by means of a few simple examples. A prototype expression compiler written to test these concepts is described in Section 3. Section 4 describes the architecture of an exemplary μFPU. The compiler and μFPU are evaluated on a number of benchmark programs in Section 5.

2 Micro-Optimizations

This section illustrates micro-optimizations by means of examples given in μFP assembly code (see Section 4). The code for a single add (`A = B + C`) and a single multiply (`A = B * C`) are shown below. The subtract operation is similar to add. The optimizations start from concatenations of these sequences and perform transformations to reduce the number of micro-operations.

```
                ADD                                  MULTIPLY

L0:     E0  = EB  E-  EC , BNEG L1       L0:     E1  = EB  E+  EC
        M0  = MC  SHR E0                         M1  = MB  M*  MC
        M1  = M0  M+  MB , BR L2                 E2  = FF1 M1
L1:     M0  = MB  SHR- E0                        EA  = E1  E-  E2
        M1  = MB  M+  M0                         MA  = M1  SHL E2
L2:     E1  = FF1 M1
        EA  = EB  E-  E1
        MA  = M1  SHL E1
```

In this section optimizations will be evaluated by comparing the path lengths of the optimized and unoptimized μFP code. Timings for different micro-operations will be discussed in Section 4.

Three instructions, at least half the total, in each sequence are used to normalize the result. Many of the optimizations described below are methods to eliminate unneccessary normalizations.

Automatic Block Exponent

The alignment operations of cascaded additions can be simplified if the largest exponent is identified and used as a block exponent for the additions. All mantissas are aligned using this exponent and added without normalization. Only the final sum is normalized.

The following code shows an application of this technique to the expression (`T0 = A + B + C`). Only the code for the case where `A` has the largest exponent is shown. By eliminating the normalization and realignment of the intermediate result, this path through the sum has been reduced from 12 instructions to 9.

```
L0:     E1  = EA  E-  EB , BNEG L1
        E2  = EA  E-  EC , BNEG L2
        M1  = MB  SHR E1
        M2  = MA  M+  M1
        M3  = MC  SHR E2
        M4  = M2  M+  M3 , BR L4
        <L1 and L2 omitted for clarity>
L4:     E4  = FF1 M1
        ET0 = EA  E-  E4
        MT0 = M4  SHL E4
```

The use of automatic block exponent requires that extra mantissa bits to the left of the binary point be maintained in case the adds result in an increased exponent. If n adds are performed in sequence, $\log_2 n$ extra bits must be maintained.

In some cases, the use of an automatic block exponent can increase rounding errors. In the above example, if `A` $\approx$ `-B` and `|C|` << `|A|`, the intermediate result is badly undernormalized and valuable bits of `C` will be lost when it is aligned with the original exponent. The effect is the same as if the addition were performed in the order `(A + C + B)`. This technique treats floating point addition as if it were associative and commutative and has the same effect as reordering the additions to give the largest possible rounding error.

Even with these limitations, automatic block exponent is a very effective optimization. Many computations include long sequences of adds (e.g., dot products) where operand ordering is not critical. In these cases, the use of a block exponent reduces the path length by from $6n$ to $3n + 3$, a savings of 50%!

Shift Combining

Shift combining is an alternative to automatic block exponent that can be used in cases where the order of the operations must be preserved. When adding three or more floating point numbers, redundant shifts may be performed when a mantissa is shifted left for normalization and then immediately shifted right for alignment. To recognize redundant shifts, the mantissa left shift in the first add is moved below the branch of the second add. This requires copying the shift into both paths of the branch. The shift will be eliminated in one of the two paths.

The following code fragment, taken from the compilation of `A + B + C`, illustrates this technique. The fragment begins after the B and C mantissas have already been aligned and added. It ends after the final mantissa sum is computed but before the normalization.

```
BEFORE OPTIMIZATION                    AFTER OPTIMIZATION

L2:     E1  = FF1 M1                   L2:     E1  = FF1 M1
        ET0 = EB  E-  E1                       ET0 = EB  E-  E1
        MT0 = M1  SHL E1                       E2  = EA  E-  ET0, BNEG L3
        E2  = EA  E-  ET0, BNEG L3             E3  = EA  E-  EB
        M2  = MT0 SHR E2                       M2  = M1  SHR E3
        M3  = M2  M+  MA,  BR L4               M3  = M2  M+  MA,  BR L4
L3:     M2  = MA SHR- E2               L3:     MT0 = M1  SHL E1,
        M3  = MT0  M+  M2                      M2  = MA  SHR- E2
                                               M3  = MT0 M+  M2
```

The left shift of `M1` has been pushed below the branch on (`EA >= ET0`). If the branch is not taken, the shift is combined with the alignment right shift. An additional exponent subtract is required to calculate the shift count. If the branch is taken, the shifts operate on different mantissas and cannot be combined. The path length of the optimized code is unchanged, but an expensive mantissa shift is replaced with an inexpensive exponent subtract.

Post Multiply Normalization

A multiply operation can denormalize its result by at most one bit position. If a few extra guard bits to the right of the mantissa are maintained, the results of multiplication can be used without normalization with no loss of accuracy. Only the final result must be normalized. For example, the code for `A * B * C` is shown below.

```
L0:     E1  = EB  E+  EC
        M1  = MB  M*  MC
        E2  = E1  E+  EA
        M2  = MA  M*  M1
        E3  = FF1 M2
        ET0 = E3  E-  E2
        MT0 = M2  SHL E3
```

This optimization also handles the ubiquitous case of multiply-add. If a multiply is followed by an add, its normalization can be eliminated as the final result will be normalized by the add.

For a sequence of multiplies, this optimization reduced the number of intructions from $5n$ to $2n + 3$, a savings of 60%. The savings in terms of time is somewhat less since the mantissa multiply M* is an extremely costly operation.

Conventional Optimizations

Decomposing floating-point operations exposes the resulting micro-operations to conventional compiler optimizations such as *constant folding*, *common* subexpression elimination, and dead code elimination. Consider for example, the expression (A + B)*(A - B). When reduced to micro-operations the alignment of A and B can be recognized as a common subexpression and eliminated. The optimization reduces the path length from 17 to 15, a 12% improvement. A source level compiler can find no common subexpressions and will perform the alignment twice.

Scheduling

More efficient use of floating point hardware can be made by scheduling the micro-operations of an entire floating-point expresion as a unit rather than scheduling each add or multiply separately. The μops of one floating point operations can be used to fill idle cycles in the evaluation of other floating point operations even if there are dependencies between the two operations.

Consider for example the case of a multiply-add (A * B + C). A reservation table for this operation is shown below. Once the exponent addition for the multiply is completed (A), the exponent subtract for the add may be performed (C). If EA + EB > EC, the alignment shift for the add (D) may also be performed in parallel with the multiply (B). In a conventional floating point unit, the multiply has to complete before any part of the add can be performed.

```
Unit    1 2 3 4 5 6 7 8 9 10
M *     B B B B
M +           E E
M SH        D D         H H
M FF1             F F
E +/-   A C             G
```

```
A: E1  = EA  E+  EB
B: M1  = MA  M*  MB
C: E2  = E1  E-  EC, BNEG L1
D: M2  = MC  SHR E2
E: M3  = M1  M+  M2, BR   L2
```

3 The Micro-Optimizer

An experimental micro-optimizer has been implemented to evaluate the optimizations described above. The program accepts a restricted LISP expression as input and produces optimized μFPU assembly code as output.

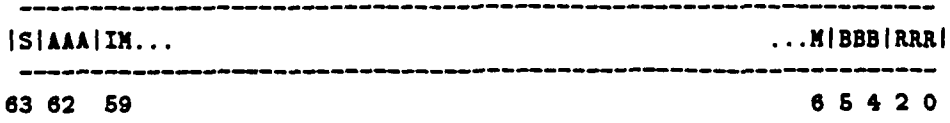
The compilation is performed in the following steps

1. The expression is compiled into standard three address *macro* floating-point assembly code.
2. A data flow graph is constructed and used to recognize (1) sequences of cascaded additions and (2) non-terminal multiplies.
3. With the aid of the data flow graph, the *macro* assembly code is translated into μFP code. Automatic block exponent and post multiply normalization optmizations are performed during this step.
4. Shift combining is performed by checking each shift to determine if its result is used as input to another shift.
5. A control flow graph is constructed and each statement is labeled with an identifier specifying the paths that pass through that statement.
6. With the aid of the control flow graph, common subexpression elimination is performed. Expressions are eliminated outside of basic blocks if they are labeled with the same path identifier.
7. The optimized μFP code is scheduled into horizontal microinstructions using a greedy algorithm that schedules an operation as soon as its inputs and required resources are available.

4 A Micro Floating Point Unit

A micro floating point unit (μFP) is required to efficiently execute the code produced by the micro-optimizer. Micro-optimization reduces floating point operations to their constituent integer operations; however an integer processor does not support features such as *sticky* bits that are required to round according to existing standards [2]. This section describes the architecture of a μFP suitable to execute the code described above. The purpose of this design is to serve as a basis for the evaluation made in Section 5. This description is a paper design, no μFPU has been constructed.

The μFP contains a 31-word by 12-bit exponent register file, and a 31-word by 64-bit mantissa file. Each register file has two read ports and a single write port. The exponent registers contain 12-bit 2's complement numbers. These numbers are converted to/from offset format during load and store operations. The mantissa registers have the format shown below. A 55-bit mantissa (M) includes the implied bit (I), and sign bit S. The mantissa is protected above by three A bits and below by three B bits as well as the standard guard, round, and sticky bits (R).



The A bits allow up to four aligned mantissa additions to be performed before normalizing the result. The possible one-bit overflows are accumulated in the A bits for later normalization. The B bits allow up to four multiplies to be performed before normalizing. The bits that shift off to the right because of the possible one-bit denormalization are accumulated in the B bits and the guard bit.

The exponent and mantissa data paths are shown in Figure 1. The exponent path has an adder/subtractor and can receive data from the find-first-one (FF1) unit in the mantissa path. The mantissa path includes a multiplier, an adder, a shifter, and a find-first-one unit The multiplier, adder, shifter, and FF1 unit are pipelined with latencies of 4,2,2, and 2 (see below). The shifter sets the sticky bit of the result if any ones are discarded from the right side of the operand. The adder uses the round and sticky bits to round each addition. The multiplier both produces the rounding bits and uses them to round the result.

There are two crossovers between the exponent and mantissa data paths. The mantissa shift is controlled by an exponent shift count, and the find-first-one unit takes a mantissa as input and produces an exponent result.

The clock cycle is determined by the time required for a 12-bit exponent add ($\approx$ 15ns in a 1μ CMOS technology). Assuming a carry lookahead adder and a Wallace-tree multiplier [4], times for mantissa multiply, add, shift, and find-first-one are estimated to be 4, 2, 2, and 2 cycles respectively. A register read or write takes one clock cycle, and a register can be read in the same cycle it is written. There is full bypassing under compiler control (no comparators).

The format of a μFP instruction is shown below. Each instruction specifies sources and destinations for the mantissa and exponent register files, the exponent and mantissa operations, and a branch specifier. Specifying a register address of all ones (0x1F) selects a bypass from the result bus. Branches have no delay if not taken and a one cycle delay if taken.

Instruction Format:

EA	EB	EC	MA	MB	MC	EOP	MOP	BOP	BDST

The units perform the following operations. Each unit also has a NOP operation.

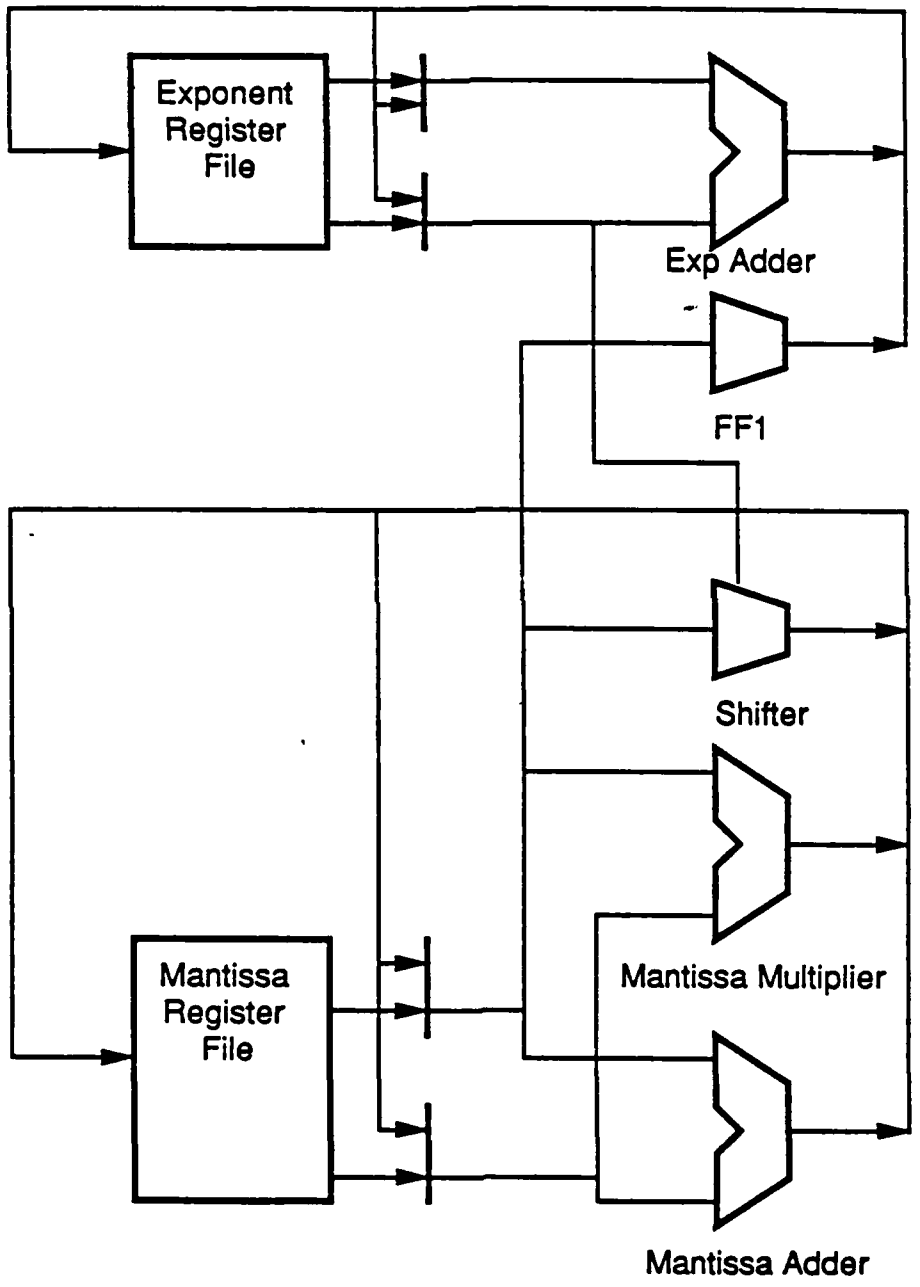


Figure 1: μFPU Data Paths

Exponent OPs	
E+, E-	Exponent add/subtract (EC <- EA op EB).
FF1	Returns the shift required to normalize mantissa MA (EC <- FF1 MA). In the range [-3,57]. Returns the largest positive number if no ones are found.
LDE, STE	Load or store exponent as an integer.
Mantissa OPs	
M+, M-, M*	Mantissa add, subtract, and multiply (MC <- MA op MB).
SHR, SHL	Mantissa right and left shift (MC <- MA >> EA) or (MC <- MA << EA). A negative exponent shifts in the opposite direction.
ABS, NEG	Zeros and complements the mantissa sign bit.
LDM, STM	Load or store mantissa as an integer.
LDF, STF	Load or store mantissa and exponent formatted as a standard floating point number.
Branch OPs BR	Unconditional branch.
BNEG	Branch on exponent negative (EC < 0).
Bcond	Branch on exponent and mantissa compare (EA, MA) relop (EB, MB).

This instruction set is the minimum required to perform the evaluation in the next section. In certain applications additional instructions would be useful. For example, if divides were used frequently a mantissa divide M/ could be realized with an SRT divide array. If divides are less frequent, a reciprocal approximation can be programmed using the instructions above.

This instruction set is intended to complement a simple integer instruction set [7] [1] [6]. For operations such as reciprocal and square root that are often performed using Newton's method, there is no need to implement an initial approximation lookup table in the μFPU. These tables can be kept in main memory and accessed using integer instructions. By exposing the algorithms for reciprocal, square root, and other floating-point functions, the compiler can perform optimizations that are not possible if these functions are hidden in microcode.

5 Evaluation

To evaluate micro-optimizations, the μFPU described in Section 4 is compared against a conventional floating point unit (cFPU) with the same micro-operation times and register file bandwidth. The two units were compared on a series of benchmark expressions. For each expression and each unit, the total number of micro operations *operation count* and the total number of clock cycles required *time* to execute the longest path through the expression is measured.

The following assumptions are made:

- The two units have identical clock rates and micro operation times.

- Each cycle, each unit can read two mantissas and two exponents and write one mantissa and one exponent.
- All units are pipelined and can accept a new input each cycle.
- Branches have no delay if not taken and a delay of one if taken.
- Common subexpression elimination is performed on the **macro** floating point operations for both units.
- The operations on each unit were scheduled using a greedy algorithm.

The benchmarks are summarized in the following table:

Benchmark	Description	*	+
1	(+ (* a a) (* b b)))	2	1
2	(+ a b c d))	0	3
3	(* a b c d))	3	0
4	Simple MOSFET Equation	3	3
5	3-D dot product	3	2
6	Acceleration Calculation	8	7
7	Magnitude of Butterfly	8	9
8	8 Tap FIR Filter	8	7

The operation counts and times for the twelve cases are tabulated below along with total lengths and times for the two units.

Over the six benchmarks, micro-optimizations resulted in a 33 % reduction in operation count and a 40 % reduction in time. The reductions are largest for large expressions with long sequences of adds or multiplies.

Expressions with a great deal of internal parallelism give a smaller reduction in execution time. The parallelism in these expressions can keep a conventional floating point pipeline very busy reducing the advantage gained by independently scheduling micro-operations. For example, the FFT butterfly operation (benchmark 7) calculates the real and imaginary components of its two outputs in parallel. A pipelined FPU can execute these four calculations in parallel. Because the μFPU consumes register bandwidth handling intermediate results, it cannot initiate operations as quickly. Because of the register bandwidth bottleneck, this benchmark has a typical reduction in operation count (30%), but only a 25% reduction in execution time.

All benchmarks other than number 7 show a greater improvement in execution time than in operation count. This data suggests that register bandwidth is not an issue for most scalar expressions. The two units were compared with identical and realistic register file bandwidth. Data dependencies prevent the conventional FPU from exploiting all of this bandwidth. If memory bandwidth is equal to register bandwidth, a conventional FPU will outperform a μFPU on vector operations. The conventional unit can start an operation each cycle while the

μFPU will use some register bandwidth for intermediate results. When register bandwidth is at least twice memory bandwidth, the μFPU becomes competitive even on vector operations.

Operation Count			
Benchmark	cFPU	μFPU	% Reduction
1	16	10	38
2	18	15	17
3	15	9	40
4	33	26	21
5	27	17	37
6	82	56	32
7	94	67	29
8	82	47	43
TOTAL	367	247	33

Time (cycles)			
Benchmark	cFPU	μFPU	% Reduction
1	21	13	38
2	30	19	37
3	30	16	47
4	50	31	38
5	32	17	47
6	94	52	45
7	73	55	25
8	87	47	46
TOTAL	417	250	40

6 Conclusion

A technique for micro-optimizing floating-point expressions has been described. Micro-optimization involves reducing floating-point expressions to their constituent micro-operations and optimizing the resulting sequence. By exposing the micro-operations to the compiler many redundant operations can be eliminated. Scheduling of individual micro-operations allows dependent macro operations to be partially overlapped.

An evaluation of micro-optimization shows that it reduces operation count by 33 % and execution time by 40 % compared to conventional floating-point execution. The operation count reduction is largely due to the elimination of unecessary normalization operations. Elimination of common exponent subexpressions contributes a small amount. The improvement in execution time is due to the elimination of these operations and the increased overlap of operations resulting from scheduling micro-operations separately. In some cases exponent calculations are scheduled in such a manner that the execution time is entirely due to mantissa calculations.

A micro floating-point unit is required to execute these floating-point micro-operations. Although they are integer operations, appropriate word lengths and support for rounding are required to maintain accuracy. Also, separate mantissa and exponent paths are required to give performance competitive with conventional floating point units.

A μFPU breaks the pipeline of a conventional floating-point unit into separately schedulable function units. The additional scheduling flexibility can be exploited through micro-optimization. The penalty for this separation is potentially higher register file bandwidth, higher instruction bandwidth and increased control complexity.

The flexibility inherent in a μFPU has many advantages other than performance. For example, it can be used to gracefully support high precision floating point numbers. If provision is made in the μFPU to recover the low bits of a multiply and to link carry bits between adds, high-precision floating point arithmetic can be implemented at about the same cost as high-precision integer arithmetic.

A μFPU can also make tradeoffs between area and performance. For example, a smaller unit could be constructed that performs mantissa multiply with two or four multiply step operations. The resulting unit would be significantly smaller and would be slower only in those cases where two mantissa multiplies can be overlapped.

The work described here is an effort to integrate floating-point arithmetic into RISC computer architecture [8]. Conventional RISCs operate with a scalar and/or vector floating point unit that is operated separately from the RISC pipeline. A μFPU integrates floating point operations into the pipeline so that only one execution controller is required. Floating-point micro-operations are handled in the same manner as integer operations.

Most floating point calculations are limited by memory bandwidth rather than by arithmetic capability. By integrating floating-point and address calculation in one unit, the coupling between the FPU and the memory system can be made tighter. For example, micro-operations can be used to fill the delay slots of a delayed load. Because these operations are scheduled by the compiler, no time and bandwidth is lost synchronizing data arrival with a separately scheduled floating point pipeline.

Much work remains to be done on micro-optimizations. Extending the expression compiler of Section 3 into a full compiler will create opportunities for additional optimization. For example, loops that iterate over arrays accumulating a running sum can be optimized with a technique similar to automatic block exponent. Other optimizations become possible if the compiler is extended to infer the signs and relative magnitudes of some variables. If the two inputs to a mantissa add can be shown to have the same sign, the result will not be denormalized (it may overflow one bit), and the sign of the result can be inferred. If exponent values can be inferred or computed early, block exponents can be applied across large expressions. If the relative magnitudes of exponents can be inferred, branches on exponent comparison can be eliminated.

Floating point numbers are popular because they free the programmer from the tedious task of scaling integers. Scaling need not be performed entirely at run-time by hardware, however. A suitable division of effort between a micro-optimizing compiler and hardware with some *primitive* support for floating point can result in substantial performance improvement.

Acknowledgement

The work presented here has benefited from discussions with Anant Agarwal, Tom Knight, Scott Wills, and Steve Ward.

References

[1] AMD, *AMD 29000 User's Manual*, 1987.

[2] ANSI/IEEE Standard 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic.*

[3] Colwell, R.P., et.al., "A VLIW Architecture for a Trace Scheduling Compiler," *IEEE Trans. Computers*, C-37(8), August 1988, pp. 967-979.

[4] Hwang, K. , *Computer Arithmetic: Principles, Architecture, and Design*, Wiley, 1979.

[5] Magenheimer, et.al., "Integer Multiplication and Division on the HP Precision Architecture," *IEEE Trans. Computers*, C-37(8), August 1988, pp. 980-990.

[6] Motorola, *MC88100 32-bit Third-Generation RISC Microprocessor: Technical Summary*, Document BR588/D, 1988.

[7] Moussouris, J. et.al, "A CMOS RISC Processor with Integrated System Function," *COMPCON*, 1986, pp. 126-131.

[8] Patterson, David A., "Reduced Instruction Set Computers," *CACM*, 28(1), January 1985, pp. 8-21.

[9] Strecker, W.D., "VAX-11/780, A Virtual Address Extension to the PDP-11 Family", Proc. NCC, 1978, pp. 967-980.